

O'REILLY®

TURING

图灵程序设计丛书

第3版



PostgreSQL

即学即用

PostgreSQL: Up and Running

涵盖PostgreSQL核心概念与功能特性

[美] 瑞金娜·奥贝 利奥·徐 著
丁奇鹏 译



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS

--	--	--	--

PostgreSQL 3

$$\frac{d}{dt} \left(\frac{1}{\rho} \right) = - \frac{1}{\rho^2} \frac{d\rho}{dt}$$

□ □ □ □ □ □

ISBN 978-7-115-49966-0

[illegible][illegible][illegible][illegible]

091507240605ToBeReplacedWithUserId

11/11

O'Reilly Media, Inc. □□

10

11

10

PostgreSQL

□ □ □ □ □ □ □

10

--	--	--	--	--	--

O'Reilly Safari

□□□□

□□□

□ 1 □ □□□□

1.1 □□□□□□PostgreSQL

1.2 □□□PostgreSQL□□□

1.3 □□□□PostgreSQL

1.4 □□□□

1.4.1 psql

1.4.2 pgAdmin

1.4.3 phpPgAdmin

1.4.4 Adminer

1.5 PostgreSQL□□□□□

1.6 □□□□□PostgreSQL□□□□□□□□

1.6.1 □□□□□□

1.6.2 PostgreSQL 10□□□□□□□□

1.6.3 PostgreSQL 9.6□□□□□□□□

1.6.4 PostgreSQL 9.5□□□□□□□□

1.6.5 PostgreSQL 9.4□□□□□□□□

1.7 □□□□□□□□

1.8 □□□□□□

1.9 PostgreSQL□□□□□□□□

□ 2 □ □□□□□

2.1 □□□□

2.1.1 □□□□□□□□

2.1.2 postgresql.conf

2.1.3 pg_hba.conf

2.2 □□□□

□□□□□□□□□□□□

2.3 □□

- 2.3.1 创建数据库
 - 2.3.2 创建表
- 2.4 管理database
 - 2.4.1 创建表
 - 2.4.2 schema管理
- 2.5 用户管理
 - 2.5.1 创建用户
 - 2.5.2 用户组
 - 2.5.3 GRANT
 - 2.5.4 角色
 - 2.5.5 PostgreSQL用户管理
- 2.6 备份与恢复
 - 2.6.1 备份
 - 2.6.2 恢复
- 2.7 性能调优
 - 2.7.1 使用pg_dump备份数据库
 - 2.7.2 使用pg_dumpall备份数据库
 - 2.7.3 性能调优
- 2.8 安全与审计
 - 2.8.1 安全
 - 2.8.2 审计
- 2.9 其他
 - 2.9.1 使用PostgreSQL
 - 2.9.2 使用PostgreSQL
 - 2.9.3 使用shared_buffers
 - 2.9.4 使用PostgreSQL

3 psql

- 3.1 安装
- 3.2 psql

3.3 安装psql数据库

3.3.1 安装psql数据库

3.3.2 安装数据库

3.3.3 安装数据库

3.3.4 安装

3.3.5 安装数据库

3.4 psql数据库

3.4.1 安装shell

3.4.2 安装watch数据库

3.4.3 安装数据库

3.4.4 安装

3.4.5 安装SQL

3.5 安装psql数据库

3.5.1 安装psql数据库

3.5.2 安装psql数据库

3.5.3 安装数据库

3.6 安装psql数据库

4 pgAdmin

4.1 pgAdmin

4.1.1 安装

4.1.2 安装PostgreSQL

4.1.3 pgAdmin

4.2 pgAdmin

4.2.1 安装SQL

4.2.2 pgAdmin3

4.2.3 pgAdmin3 postgresql.conf pg_hba.conf

5

4.2.4 安装数据库

4.2.5 安装数据库

- 4.2.6 数据库
- 4.3 pgScript数据库
- 4.4 数据库数据库数据库
- 4.5 数据库pgAgent数据库数据库
 - 4.5.1 数据库pgAgent
 - 4.5.2 数据库数据库
 - 4.5.3 数据库数据库pgAgent数据库数据库

第 5 章 数据库

- 5.1 数据库
 - 5.1.1 serial数据库
 - 5.1.2 数据库数据库数据库
- 5.2 数据库
 - 5.2.1 数据库
 - 5.2.2 数据库数据库数据库数据库数据库数据库
 - 5.2.3 数据库数据库数据库
- 5.3 数据库
 - 5.3.1 数据库
 - 5.3.2 数据库数据库数据库数据库数据库
- 5.4 数据库
 - 5.4.1 数据库数据库
 - 5.4.2 数据库数据库数据库数据库
 - 5.4.3 数据库数据库数据库
 - 5.4.4 数据库数据库数据库
 - 5.4.5 数据库数据库数据库
- 5.5 数据库
 - 5.5.1 数据库数据库数据库数据库
 - 5.5.2 数据库数据库数据库数据库
 - 5.5.3 数据库数据库数据库
 - 5.5.4 数据库数据库数据库数据库数据库

5.5.5 数据库数据库

5.6 JSON数据库

5.6.1 数据库JSON

5.6.2 数据库JSON

5.6.3 数据库JSON

5.6.4 JSON数据库数据库jsonb

5.6.5 数据库JSONB数据库

5.7 XML数据库

5.7.1 数据库XML

5.7.2 数据库XML

5.8 数据库

5.8.1 FTS数据库

5.8.2 TSVector数据库

5.8.3 TSQueries数据库

5.8.4 数据库

5.8.5 数据库

5.8.6 数据库

5.8.7 数据库JSONJSONB数据库

5.9 数据库数据库

5.9.1 数据库数据库数据库

5.9.2 数据库

5.9.3 数据库

5.9.4 数据库数据库数据库

6 数据库

6.1 数据库

6.1.1 数据库

6.1.2 数据库

6.1.3 数据库

6.1.4 数据库

6.1.5 TYPE OF

6.2

6.2.1

6.2.2

6.2.3 check

6.2.4

6.3

6.3.1 PostgreSQL

6.3.2

6.3.3

6.3.4

6.3.5

7 PostgreSQL SQL

7.1

7.1.1

7.1.2

7.1.3

7.2 PostgreSQL SQL

7.2.1 DISTINCT ON

7.2.2 LIMIT OFFSET

7.2.3

7.2.4

7.2.5 ILIKE

7.2.6 ANY

7.2.7

7.2.8 DELETE UPDATE INSERT

7.2.9 DELETE USING

7.2.10

7.2.11 UPSERT INSERT UPDATE

- 7.2.12 `ORDER BY`
- 7.2.13 `ORDER BY` `$`
- 7.2.14 `DO`
- 7.2.15 `ORDER BY` `FILTER`
- 7.2.16 `ORDER BY` `ORDER BY`
- 7.3 `ORDER BY`
 - 7.3.1 `PARTITION BY`
 - 7.3.2 `ORDER BY`
- 7.4 `CTE`
 - 7.4.1 `CTE`
 - 7.4.2 `CTE`
 - 7.4.3 `CTE`
- 7.5 `LATERAL`
- 7.6 `WITH ORDINALITY`
- 7.7 `GROUPING SETS` `CUBE` `ROLLUP`

8 `ORDER BY`

- 8.1 `PostgreSQL`
 - 8.1.1 `ORDER BY`
 - 8.1.2 `ORDER BY`
 - 8.1.3 `ORDER BY`
 - 8.1.4 `ORDER BY`
- 8.2 `SQL`
 - 8.2.1 `SQL`
 - 8.2.2 `SQL`
- 8.3 `PL/pgSQL`
 - 8.3.1 `PL/pgSQL`
 - 8.3.2 `PL/pgSQL`
- 8.4 `PL/Python`
 - `Python`

8.5 PL/V8PL/CoffeeScriptPL/LiveScript

8.5.1

8.5.2 PL/V8

8.5.3 PL/V8

9

9.1 EXPLAIN

9.1.1 EXPLAIN

9.1.2

9.1.3

9.2

9.3 SQL

9.3.1 SELECT

9.3.2 SELECT *

9.3.3 CASE

9.3.4 FilterCASE

9.4

9.4.1

9.4.2

9.4.3

9.5

9.5.1

9.5.2

9.5.3

9.5.4

9.6

10

10.1

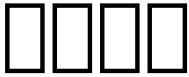
10.1.1

10.1.2

- 10.1.3 数据库用户
- 10.2 数据库
- 10.2.1 数据库
 - 10.2.2 数据库数据库数据库
 - 10.2.3 数据库
 - 10.2.4 数据库数据库数据库database
- 10.3 数据库
- 10.3.1 数据库
 - 10.3.2 数据库数据库数据库数据库
 - 10.3.3 数据库PostgreSQL数据库
 - 10.3.4 数据库ogr_fdw数据库数据库数据库
 - 10.3.5 数据库
- 附录 A PostgreSQL
- A.1 Windows数据库Linux
 - A.2 CentOS数据库Fedora数据库Red Hat数据库Scientific Linux
 - A.3 Debian数据库Ubuntu
 - A.4 FreeBSD
 - A.5 macOS
- 附录 B PostgreSQL数据库
- B.1 数据库pg_dump数据库
 - B.2 数据库数据库pg_dumpall
 - B.3 database数据库数据库pg_restore
 - B.4 数据库psql
 - B.5 数据库数据库psql

附录

附录



© 2018 by Regina Obe and Leo Hsu.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Posts & Telecom Press, 2018. Authorized translation of the English edition, 2018 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

© O'Reilly Media, Inc. 2017

2018 O'Reilly Media, Inc. — O'Reilly Media, Inc.

O'Reilly Media, Inc.

O'Reilly Media 1978 O'Reilly —“” O'Reilly

O'Reilly “” GNN Make DIY O'Reilly O'Reilly

“O'Reilly 的书籍是 IT 专业人士的必备读物”
——《O'Reilly 书籍》

“O'Reilly

“O'Reilly Radar 书籍”

——Wired

“O'Reilly 书籍是 IT 专业人士的必备读物”

——Business 2.0

“O'Reilly Conference 书籍”

——CRN

“O'Reilly 书籍是 IT 专业人士的必备读物”

——Irish Times

“Tim 书籍是 IT 专业人士的必备读物”
Yogi Berra 说：“O'Reilly 书籍是 IT 专业人士的必备读物”
Tim 书籍是 IT 专业人士的必备读物”

——Linux Journal

“

PostgreSQL 书籍是 IT 专业人士的必备读物”

PostgreSQL 书籍是 IT 专业人士的必备读物”
PostgreSQL“书籍是 IT 专业人士的必备读物”
“书籍是 IT 专业人士的必备读物”

□□□□□□ 3500 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
300 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□——□□□□

[PostgreSQL](#)

[Postgre Online Journal](#)

[PostgreSQL](#)

[PostgreSQL](#)

PostgreSQL 9.5 PostgreSQL 9.6 PostgreSQL 10

[illegible]

SQLデータベースのインストールと設定
—— PostgreSQL のインストールと設定

PostgreSQL DBA
 PostgreSQL
 PostgreSQL
 PostgreSQL

PostgreSQL PostgreSQL“”
“”
“”

0000000000000000 IT 00000000000000000000000000000000
 000“00”00000000000000000000000000000000

PostgreSQL

PostgreSQL [データベース](#) [チュートリアル](#) [HTML](#) [PDF](#) [お問い合わせ](#)

📖 PostgreSQL 📖

- Planet PostgreSQL PostgreSQL に関する記事の集まり
- PostgreSQL に関する記事の集まり
- PostgreSQL Wiki PostgreSQL に関する記事の集まり
- PostgreSQL Books PostgreSQL に関する記事の集まり
- PostGIS in Action Books PostGIS pgRouting に関する記事の集まり

□ □ □ □ □ □ □

[illegible]

```
function (
    Welcome to PostgreSQL
);
```

[illegible]

```

0000000000000000000000000000000000000000000('a','b',
'c')

```

PostgreSQL is SQL compatible database system

[illegible]

```

Linux\Windows
Linux / Windows \ PostgreSQL
Windows Linux / Windows \
/postgresql_book/somefile.csv
Linux Windows
C:/postgresql_book/somefile.csv

```

□ □ □ □

□ □ □ □ □ □ □ □ □ □ □ □

- □ □

□ □ □ □ □ □

- $\Theta(1)$ constant width

[illegible]

- `constant width bold`

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

- `constant width italic`

□ □



□ □ □ □ □ □ □ □ □ □



□ □ □ □ □ □ □ □ □ □

□ □ □ □ □ □

http://www.postgresqlonline.com/downloads/postgresql_book_3e.zip ☐☐☐

ISBN 978-1-491-96341-8 “PostgreSQL : Up and Running , Third Edition by Regina Obe and Leo Hsu (O'Reilly). Copyright 2018 Regina Obe and Leo Hsu, 978-1-491-96341-8.”

O'Reilly Safari



250
O'Reilly Media, Harvard Business Review, Prentice Hall
Professional, Addison-Wesley Professional, Microsoft Press,
Sams, Que, Peachpit Press, Adobe, Focal Press, Cisco Press,
John Wiley & Sons, Syngress, Morgan Kaufmann, IBM
Redbooks, Packt, Adobe Press, FT Press, Apress, Manning,
New Riders, McGraw-Hill, Jones & Bartlett, Course
Technology.

□□□□□□□□□□ <http://oreilly.com/safari> □

□□□□

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

111

□□□□□□□□□□ 2 □□□□□ C □ 807 □□100035□

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □

bookquestions@oreilly.com

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □

<https://www.oreilly.com/catalog/errata.csp?>

isbn=0636920052715 ☐☐¹

¹ <http://www.ituring.com.cn/book/2460> 请读者——

□□□□□□□□ <https://www.oreilly.com/catalog/errata.csp?>

isbn=0636920052715

Ir@pcorp.us

bookquestions@oreilly.com

☐ O'Reilly ☐

<http://www.oreilly.com>

Facebook <http://facebook.com/oreilly>

Twitter <http://twitter.com/oreillymedia>

看看 YouTube 频道 <http://www.youtube.com/oreillymedia>

看看

扫描二维码



第 1 章 数据库

PostgreSQL 是一个开源的数据库系统，它支持多种数据类型和复杂的查询。它被广泛使用，并且具有强大的社区支持。本章将介绍 PostgreSQL 的基本概念和用法。

本章将介绍 PostgreSQL 的基本概念和用法。PostgreSQL 是一个开源的数据库系统，它支持多种数据类型和复杂的查询。它被广泛使用，并且具有强大的社区支持。本章将介绍 PostgreSQL 的基本概念和用法。

1.1 数据库 PostgreSQL

PostgreSQL 是一个开源的数据库系统，它支持多种数据类型和复杂的查询。它被广泛使用，并且具有强大的社区支持。本章将介绍 PostgreSQL 的基本概念和用法。

```

##### PostgreSQL #####
##### SQL #####
#####
####

```

データベース NoSQL データベース PostgreSQL データベース ITree データベース PostgreSQL データベース hstore データベース JSON データベース JSONB データベース MongoDB データベース PostgreSQL データベース “NoSQL” データベース NoSQL データベース

Postgres95 PostgreSQL PostgreSQL
 1986¹ PostgreSQL
 Linux Unix Windows Mac

1 1986 Stonebraker 1988
Postgres 1989 6 1986 PostgreSQL
——

[illegible]

```

XXXXXXXXXXXXXXXXXXXXXPostgreSQLXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXX

```

1.2 PostgreSQL

PostgreSQL “ ”

PostgreSQL

☐ PostgreSQL ☐
☐ PostgreSQL ☐
☐ SQLite ☐
☐ FireBird ☐ / ☐

データベースの種類は PostgreSQL、Redis、Memcache、PostgreSQL、PostgreSQL、SQLite など

PostgreSQL、MySQL、Web、MySQL、SQL、select、join、MySQL、PaaS、DBaaS、PostgreSQL、Heroku、Engine Yard、RedHat OpenShift、Amazon RDS for PostgreSQL、Google Cloud SQL for PostgreSQL、Amazon Aurora for PostgreSQL、Microsoft Azure for PostgreSQL

1.3 PostgreSQL

PostgreSQL、Redis、Memcache、PostgreSQL、PostgreSQL、SQLite など

PostgreSQL、PostgreSQL、A

1.4

PostgreSQL、psql、pgAdmin、phpPgAdmin、Adminer、PostgreSQL、PostgreSQL、Adminer、PostgreSQL、SQLite、MySQL、SQL Server、Oracle

1.4.1 psql

psql PostgreSQL psql
B.4 HTML psql PostgreSQL
shell psql

1.4.2 pgAdmin

pgAdmin PostgreSQL PostgreSQL
PostgreSQL pgAdmin PostgreSQL
PostgreSQL

Linux pgAdmin

pgAdmin 4 1.5 PostgreSQL 9.6 Python 2.7.12
pgAdmin 4 1.5 PostgreSQL 9.6 Python 2.7.12

□ 1-1 □ pgAdmin4 □□□□□□□□



pgAdmin 4

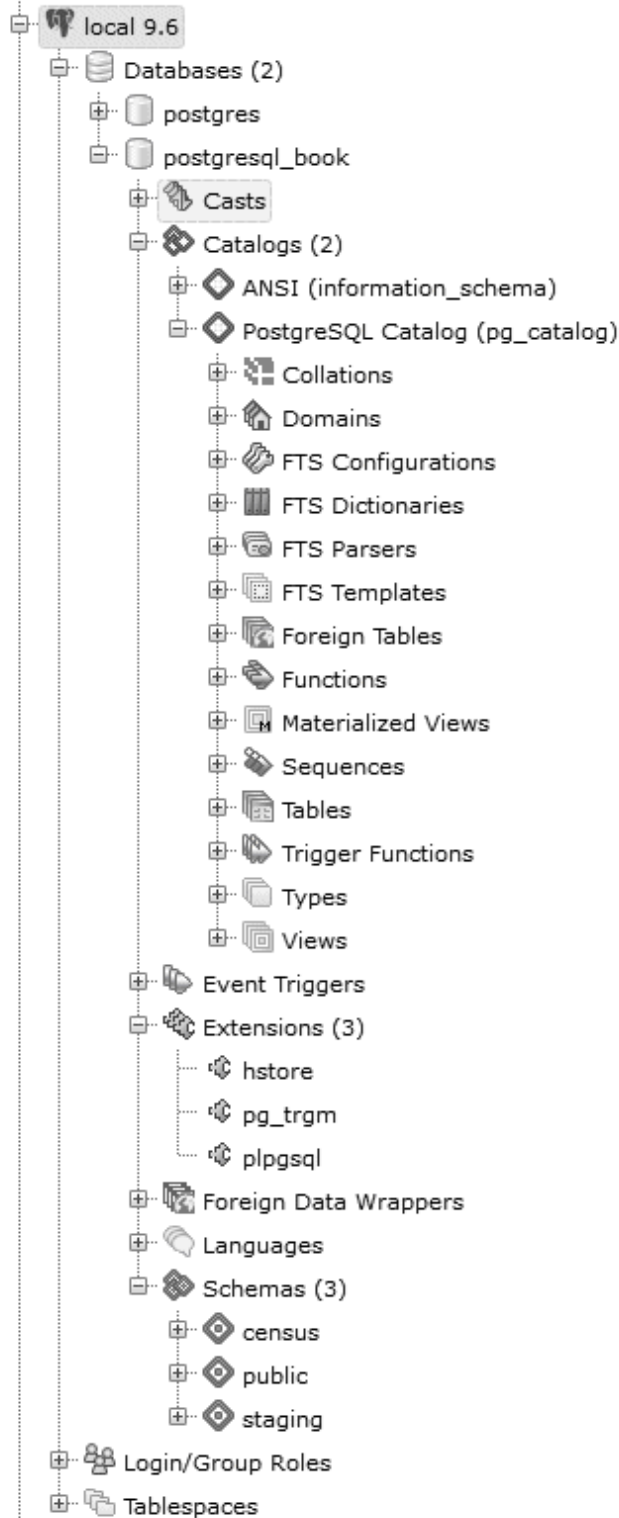
File ▾

Object ▾

Tools ▾

Help ▾

Browser



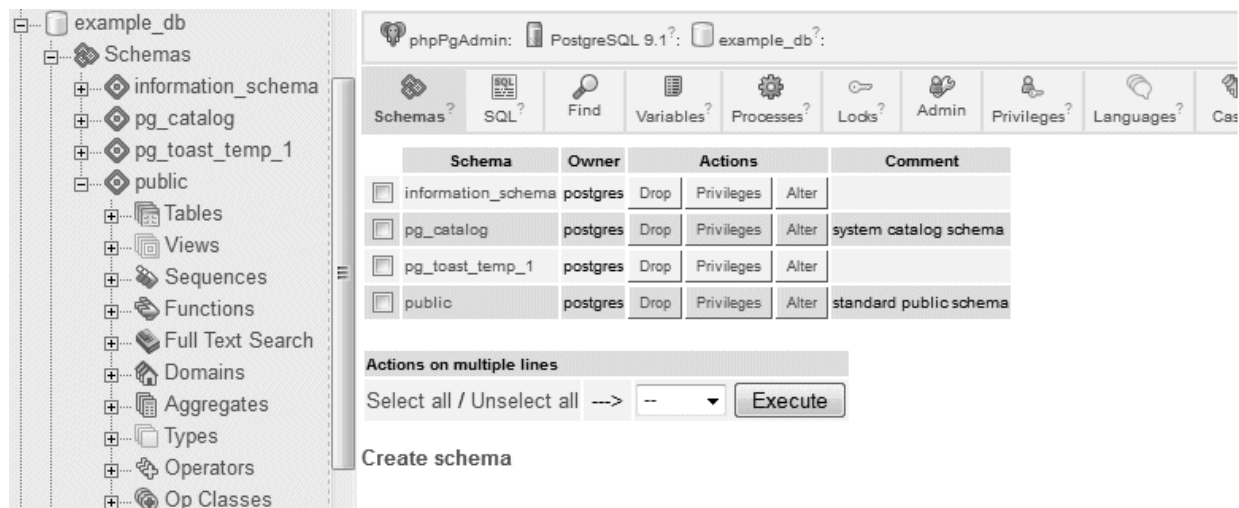
1-1 pgAdmin4

☐ PostgreSQL を管理する pgAdmin
☒ PostgreSQL のインストールと起動
☐ Microsoft SQL Server を管理する SQL Server Management Studio

```
pgAdmin3 pgAdmin4
pgAdmin3 PgAdmin3
pgAdmin4 BigSQL pgAdmin3 LTS
pgAdmin4 pgAdmin4 pgAdmin
pgAdmin3
```

1.4.3 phpPgAdmin

phpPgAdmin 数据库 Web 数据库数据库数据库 1-2 数据库数据库
MySQL 数据库 phpMyAdmin 数据库数据库数据库数据库 phpPgAdmin 数据库
数据库 PostgreSQL 数据库 schema数据库数据库数据库数据库数据库数据库数据库数据库
phpMyAdmin 数据库数据库 phpPgAdmin 数据库数据库数据库数据库



1-2 phpPgAdmin

1.4.4 Adminer

PostgreSQL Adminer PostgreSQL Adminer PHP PostgreSQL MySQL SQLite SQL Server Oracle

Adminer Adminer PHP PostgreSQL MySQL SQLite SQL Server Oracle

1-3 PostgreSQL 5432

System

PostgreSQL

Server

localhost5432

Username

postgres

Password

••••••••

Database

postgresql_book

Login

☐ Permanent login

lu fact types

fact_type_id
category
fact_subcats
short_name

lu tracts

tract_id
tract_long_id
tract_name

facts

fact_type_id
tract_id
yr
val
perc

1-3Adminer

Adminer PostgreSQL Adminer PostgreSQL Adminer schema database schema Adminer DBA pgAdmin Adminer

1.5 PostgreSQL


```
CREATE EXTENSION postgis_tiger_geocoder CASCADE;
```

PostGIS 2.2.0 安装指南

1. 安装

PostgreSQL 9.5.4 安装指南

2. 配置

PostgreSQL 9.5.4 安装指南

3. 验证

PostgreSQL 9.5.4 安装指南

4. 总结

PostgreSQL 9.5.4 安装指南

PostgreSQL 9.5.4 安装指南

pgAdmin 3 的 FDW 功能，pgAdmin 3 的 Foreign Data Wrapper 功能。

pgAdmin 3

PostgreSQL 9.0 版本，pgAdmin 3 的 Foreign Data Wrapper 功能。

pgAdmin 3 的 trigger 功能。

pgAdmin 3 的 check 功能。

PostgreSQL 9.0 版本，WITH 功能，WHEN 功能，UPDATE OF 功能，DDL 功能，DDL 功能，event trigger 功能。

catalog ⁴

⁴ catalog schema 功能，pgAdmin 3 的 catalog 功能。

catalog schema 功能，pgAdmin 3 的 catalog 功能，PostgreSQL 9.0 版本，information_schema 功能，ANSI SQL 功能，PostgreSQL 9.0 版本。

PostgreSQL 9.1.17 から 9.1.21 までの
バグ修正履歴をまとめたページを
公開しています

1.6.2 PostgreSQL 10 について

PostgreSQL 10 は 2017 年 10 月に PostgreSQL 10
が PostgreSQL 9.5 から PostgreSQL 9.6 までの
PostgreSQL 10 は PostgreSQL 11、SQLite、SQL
Server、Oracle などと

PostgreSQL 10 は

です

PostgreSQL 9.4 は

です

PostgreSQL は PostgreSQL
データベースは
database は
PostgreSQL

JSON と JSONB

to_tsvector は
JSON と JSONB は key value
ts_headline は JSON と JSONB
5.8.7

ANSI XMLTABLE

XMLTABLE 関数は XML 関数と組み合わせて使用されます。Oracle と IBM DB2 の関数集は 5-41

FDW 関数

FDW API 関数 COUNT(*) と SUM(*) は、Postgres_fdw 関数を使用して PostgreSQL の関数集に追加されます。

関数集

PostgreSQL 10 の PARTITION BY 関数は、PostgreSQL 6.1.3 の関数集に追加されます。

関数集

関数集

関数集

CREATE STATISTICS 関数は 9-18

関数 IDENTITY 関数

関数 IDENTITY 関数は 6-2

1.6.3 PostgreSQL 9.6 関数集

PostgreSQL 9.6 は 2016 年 9 月 9.x 関数集

関数集

9.6 関数 PostgreSQL 9.6 関数 PostgreSQL 関数集

join 9.4

9.6 5.8

psql \gexec

SQL 3.4.5

postgres_fdw

“Directly Modify Foreign Tables”

FDW

postgres_fdw FDW join

1.6.4 PostgreSQL 9.5

PostgreSQL 9.5 2016 1

IMPORT FOREIGN SCHEMA 10.3 10.3.3

データベースのバージョンアップ時に、データベースのスキーマを変更する必要がある。9.5 のバージョンアップ時に、ALTER TABLE ... SET UNLOGGED を使用して、データベースのスキーマを変更する必要がある。

array_agg の使用

array_agg は、データベースのバージョンアップ時に、9.5 のバージョンアップ時に、データベースのスキーマを変更する必要がある。5-17

BRIN の使用

BRIN (block range index) は、データベースのバージョンアップ時に、B-ツリー GIN と比較して、データベースのスキーマを変更する必要がある。6.3

GROUPING SETS、ROLLUP、CUBE の使用

データベースのバージョンアップ時に、7.7

データベース

データベースのバージョンアップ時に、データベースのスキーマを変更する必要がある。"データベース" GiST

"データベース" の使用 UPSERT

9.5 のバージョンアップ時に、データベースのスキーマを変更する必要がある。9.5 のバージョンアップ時に、7.2.11

データベース

データベースのバージョンアップ時に、SELECT ... FOR UPDATE を使用して、データベースのスキーマを変更する必要がある。9.5 のバージョンアップ時に、9.5 のバージョンアップ時に、SKIP LOCKED を使用して、データベースのスキーマを変更する必要がある。

データベース

GIN 索引

GIN 索引是 PostgreSQL 9.5 引入的，它支持对 hstore 和 jsonb 数据类型进行索引。B-树索引是 PostgreSQL 9.5 之前唯一支持的索引类型。GIN 索引在 PostgreSQL 9.5 中被引入，作为 B-树索引的替代品。GIN 索引的文档标题是“GIN as a Substitute for Bitmap Indexes”。

JSON 索引

PostgreSQL 9.5 引入了 json_build_array、json_build_object、json_object、json_to_record 和 json_to_recordset 函数。

这些函数用于构建和查询 JSON 数据。

要将表移动到新的表空间，可以使用 ALTER TABLESPACE 语句。

例如，将表移动到新的表空间：

ordinality 是 ANSI SQL 标准的一部分，用于指定表中的行顺序。hstore 索引是 PostgreSQL 9.5 引入的，用于对 hstore 数据类型进行索引。

```
SELECT ordinality, key, value
FROM each('breed=>pug,cuteness=>high'::hstore) WITH ordinality;
```

SQL 索引

ALTER system SET ... 用于设置 PostgreSQL 的配置文件 postgresql.conf 中的系统参数。PostgreSQL 2.1.2 版本引入了对 JSON 数据类型的支持。

索引是数据库的重要组成部分，用于提高查询效率。

9.4 索引

- .NET [.NET Npgsql](#) Mono [Npgsql](#) [Entity Framework](#) Mono [.NET](#)
- ODBC [Access](#) [Excel](#) ODBC PostgreSQL PostgreSQL ODBC 32 64
- LibreOffice/OpenOffice [LibreOffice 3.5](#) PostgreSQL 3.5 OpenOffice JDBC SDBC “OO Base and PostgreSQL”
- Python [Python](#) PostgreSQL [psycopg2](#) [Python](#) [Django](#) PostgreSQL - [ORM](#) [SQLAlchemy](#) [Multicorn](#)
- Ruby [Ruby](#) [rubygems pg](#)
- Perl [Perl](#) [DBI](#) [DBD::Pg](#) [CPAN](#) [DBD::PgPP](#)
- Node.js [Node.js](#) [JavaScript](#) PostgreSQL [Node Postgres](#) [libpq](#) [JavaScript](#) [Node-DBI](#)

1.8 ☐☐☐☐☐☐

PostgreSQL 数据库系统是一个开源的数据库系统，它支持多种数据库引擎，如 PostgreSQL、PGSQL-General、PostgreSQL 等。PostgreSQL 是一个开源的数据库系统，它支持多种数据库引擎，如 PostgreSQL、PGSQL-General、PostgreSQL 等。PostgreSQL 是一个开源的数据库系统，它支持多种数据库引擎，如 PostgreSQL、PGSQL-General、PostgreSQL 等。

1.9 PostgreSQL

```
PostgreSQL MIT/BSD
PostgreSQL
PostgreSQL
PostgreSQL
```


https://wiki.postgresql.org/wiki/PostgreSQL_derived_databases
s PostgreSQL PostgreSQL

Netezza PostgreSQL Redshift PostgreSQL PostgreSQL Amazon RDS for PostgreSQL Amazon Aurora for PostgreSQL PostgreSQL PostgreSQL SQL EnterpriseDB PostgreSQL Advanced Plus PostgreSQL Oracle Oracle EnterpriseDB PostgreSQL Postgres Plus Advanced Server PostgreSQL PostgreSQL

Postgre-X2 Postgres-XL GreenPlum PostgreSQL GreenPlum PostgreSQL

PostgreSQL PostgreSQL 2nd Quadrant BDR PostgreSQL PostgreSQL-XL PostgreSQL PostgreSQL

Citus PostgreSQL PostgreSQL PostgreSQL 9.5 PostgreSQL PostgreSQL

Google Google Cloud SQL for PostgreSQL beta

2

PostgreSQL database PostgreSQL PostgreSQL

データベースをインストールする

2.1 インストール

データベース PostgreSQL をインストールする

postgresql.conf

データベースをインストールする database をインストール PostgreSQL
の IP をインストールする

pg_hba.conf

データベース PostgreSQL をインストールする
の IP をインストールする

pg_ident.conf

データベースをインストールする PostgreSQL の
root PostgreSQL の
postgres をインストールする



PostgreSQL の role を user
group role を
の role を

データベース PostgreSQL をインストールする
pgAdmin の Admin Pack をインストールする
4.2.3 インストール 2-1
インストール

2-1 インストール

```
SELECT name, setting FROM pg_settings WHERE category = 'File  
Locations';
```

name	setting
------	---------

```

-----+-----
config_file      | /etc/postgresql/9.6/main/postgresql.conf
data_directory   | /var/lib/postgresql/9.6/main
external_pid_file| /var/run/postgresql/9.6-main.pid
hba_file         | /etc/postgresql/9.6/main/pg_hba.conf
ident_file       | /etc/postgresql/9.6/main/pg_ident.conf
(5 rows)

```

2.1.1 配置数据库

配置数据库时，PostgreSQL 数据库的配置文件位于 /etc/postgresql/9.6/main/ 目录下。其中，postgresql.conf 是主配置文件，pg_hba.conf 是主机基于访问控制配置文件，pg_ident.conf 是身份认证配置文件。在配置数据库时，context 参数指定了 postmaster 进程的 user 参数，该参数指定了数据库的上下文。

01. 配置数据库

配置数据库时，需要配置以下参数：

```
pg_ctl reload -D 数据库目录
```

在 RedHat Enterprise Linux、CentOS 和 Ubuntu 等 Linux 发行版中，PostgreSQL 数据库的配置文件位于 /etc/postgresql/9.5/main/ 目录下。

```
service postgresql-9.5 reload
```

在 PostgreSQL 9.5 版本中，数据库的配置文件位于 /etc/postgresql/9.5/main/ 目录下。其中，postgresql.conf 是主配置文件，pg_hba.conf 是主机基于访问控制配置文件，pg_ident.conf 是身份认证配置文件。

在配置数据库时，需要配置以下参数：

```
SELECT pg_reload_conf();
```

配置数据库时，需要配置以下参数：

02. PostgreSQL 启动

安装完 PostgreSQL 后，需要启动 PostgreSQL 服务。在 Linux/Unix 系统上，可以通过以下命令启动 PostgreSQL 服务。

PostgreSQL 9.6 版本在 Linux/Unix 系统上，可以通过以下命令启动 PostgreSQL 服务。

```
service postgresql-9.6 restart
```

在 Windows 系统上，可以通过以下命令启动 PostgreSQL 服务。

```
pg_ctl restart -D 数据目录
```

在 Windows 系统上，可以通过以下命令启动 PostgreSQL 服务。

2.1.2 postgresql.conf

postgresql.conf 是 PostgreSQL 的配置文件，用于配置数据库服务器的各种参数。该文件位于 PostgreSQL 的 bin 目录下。在 Windows 系统上，该文件的默认名称为 postgresql.conf，而在 Linux/Unix 系统上，该文件的默认名称为 postgresql.conf。

PostgreSQL 9.4 版本在 Linux/Unix 系统上，可以通过以下命令启动 PostgreSQL 服务。postgresql.conf 是 PostgreSQL 的配置文件，用于配置数据库服务器的各种参数。该文件位于 PostgreSQL 的 bin 目录下。在 Windows 系统上，该文件的默认名称为 postgresql.conf，而在 Linux/Unix 系统上，该文件的默认名称为 postgresql.conf。

01. postgresql.conf 配置

pg_settings 是 PostgreSQL 的配置文件，用于配置数据库服务器的各种参数。该文件位于 PostgreSQL 的 bin 目录下。在 Windows 系统上，该文件的默认名称为 postgresql.conf，而在 Linux/Unix 系统上，该文件的默认名称为 postgresql.conf。

2-2 配置

```

SELECT
    name,
    context ❶,
    unit ❷,
    setting, boot_val, reset_val ❸
FROM pg_settings
WHERE name IN (
    'listen_addresses', 'deadlock_timeout', 'shared_buffers',
    'effective_cache_size', 'work_mem', 'maintenance_work_mem')
ORDER BY context, name;

```

name	context	unit	setting	boot_val	reset_val
listen_addresses	postmaster		*		
localhost			*		
shared_buffers	postmaster	8kB	131584	1024	131584
deadlock_timeout	superuser	ms	1000	1000	
effective_cache_size	user	8kB	16384	16384	16384
maintenance_work_mem	user	kB	16384	16384	16384
work_mem	user	kB	5120	1024	5120

❶ context 設定の適用範囲を指定する
context 指定

context 指定 user 設定はデータベースユーザごとに適用される
データベースユーザごとに適用される user 指定
データベースユーザごとに適用される

context 指定 superuser 設定はスーパーユーザにのみ適用される
スーパーユーザにのみ適用される

context 指定 postmaster 設定は Postmaster プロセスにのみ適用される
PostgreSQL の起動時に PostgreSQL が読み込まれる


```

-----+-----+-----
effective_cache_size | E:/data96/postgresql.auto.conf| 11
| 8GB                | t
listen_addresses     | E:/data96/postgresql.conf      | 59
| *                  | t
maintenance_work_mem | E:/data96/postgresql.auto.conf| 3
| 16MB              | t
shared_buffers       | E:/data96/postgresql.conf      | 115
| 128MB             | f
shared_buffers       | E:/data96/postgresql.auto.conf| 5
| 131584            | t

```

postgresql.conf postgresql.auto.conf

listen_addresses

PostgreSQL IP localhost

 IPV6 IPV4 * IP

 PostgreSQL

port

PostgreSQL 5432

 PostgreSQL

max_connections

log_destination

stderr csvlog

 logging_collection on

PostgreSQL 9.6 默认配置下，shared_buffers 的大小是 128MB，这个值对于大多数系统来说都是偏小的，建议将其调大一些。

shared_buffers

shared_buffers 的大小应该设置为系统内存的 25% 左右，对于 8GB 的系统，建议设置为 2GB。PostgreSQL 9.6 默认配置下，shared_buffers 的大小是 128MB，这个值对于大多数系统来说都是偏小的，建议将其调大一些。

effective_cache_size

effective_cache_size 的大小应该设置为系统内存的 50% 左右，对于 8GB 的系统，建议设置为 4GB。PostgreSQL 9.6 默认配置下，effective_cache_size 的大小是 128MB，这个值对于大多数系统来说都是偏小的，建议将其调大一些。

work_mem

work_mem 的大小应该设置为系统内存的 1% 左右，对于 8GB 的系统，建议设置为 8MB。PostgreSQL 9.6 默认配置下，work_mem 的大小是 1MB，这个值对于大多数系统来说都是偏小的，建议将其调大一些。

maintenance_work_mem

maintenance_work_mem 的大小应该设置为系统内存的 1% 左右，对于 8GB 的系统，建议设置为 8MB。PostgreSQL 9.6 默认配置下，maintenance_work_mem 的大小是 1MB，这个值对于大多数系统来说都是偏小的，建议将其调大一些。

max_parallel_workers_per_gather

9.6 版本中，max_parallel_workers_per_gather 的大小应该设置为系统内存的 1% 左右，对于 8GB 的系统，建议设置为 8MB。PostgreSQL 9.6 默认配置下，max_parallel_workers_per_gather 的大小是 1，这个值对于大多数系统来说都是偏小的，建议将其调大一些。

PostgreSQL 9.6
max_worker_processes 8
worker worker

10 max_parallel_workers
worker

02. postgresql.conf

PostgreSQL 9.4 ALTER SYSTEM SQL
work_mem

```
ALTER SYSTEM SET work_mem = '500MB';
```

postgresql.conf
postgresql.auto.conf

```
SELECT pg_reload_conf();
```

postgresql.conf include include_if_exists

```
include 'XXXXX'
```

postgresql.conf

03. “postgresql.conf”

PostgreSQL `pg_log` `shared_buffers` `postmaster.pid` PostgreSQL

`shared_buffers` `postmaster.pid` PostgreSQL

2.1.3 pg_hba.conf

`pg_hba.conf` IP PostgreSQL `pg_hba.conf` 2-4

2-4 `pg_hba.conf`

```
# TYPE DATABASE USER ADDRESS METHOD
host all all 127.0.0.1/32 ident ❶
host all all ::1/128 trust ❷
host all all 192.168.54.0/24 md5 ❸
hostssl all all 0.0.0.0/0 md5 ❹

# TYPE DATABASE USER ADDRESS METHOD
# Allow replication connections from localhost,
# by a user with replication privilege. ❺
#host replication postgres 127.0.0.1/32 trust
#host replication postgres ::1/128 trust
```

❶ `ident` `trust` `md5` `peer` `password`

❷ IPv6 IPv6 `pg_hba.conf`

❸ IPv4 192.168.54.0/24 PostgreSQL

④ SSL 証明書インストール SSL 証明書インストール PostgreSQL 接続

SSL 証明書 postgresql.conf に postgresql.auto.conf にも
ssl ssl_cert_file ssl_key_file にも SSL
PostgreSQL にも ssl key にも

5 PostgreSQL のインストール

```

postgres postgres pg_hba.conf postgres
postgres postgres postgres postgres postgres postgres postgres postgres
postgres postgres postgres postgres postgres postgres postgres postgres
postgres postgres postgres postgres postgres postgres postgres postgres
postgres postgres postgres postgres postgres postgres postgres postgres +0.0.0.0/0
reject+ postgres +127.0.0.1/32 trust postgres postgres
postgres postgres

```

```
PostgreSQL 10 の pg_hba_file_rules を pg_hba.conf に書き出す
```

01. “pg_hba.conf”

```

#####
##### postgres ##### pg_hba.conf #####
#####
##### pg_log #####
#####
#####
#####

```

02. □□□□□□

PostgreSQL 数据库认证方式有 trust peer ident md5 password reject 这几种，在 pg_hba.conf 文件中，认证方式有 gss sspi ldap radius cert pam 这几种，PostgreSQL 数据库认证方式有 trust peer ident md5 password reject 这几种，在 pg_hba.conf 文件中，认证方式有 gss sspi ldap radius cert pam 这几种。

gss radius ldap pam

database
database
PostgreSQL pg_hba.conf

2.2

SQL

“” SQL

(1) ID

```
SELECT * FROM pg_stat_activity;
```

pg_stat_activity username
database datname
ID

(2) 1234

```
SELECT pg_cancel_backend(1234);
```

データベース

(3) テーブル

```
SELECT pg_terminate_backend(1234);
```

データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。

PostgreSQL では `SELECT` ステートメントで `pg_terminate_backend` と `pg_cancel_backend` を使用して、データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。

```
SELECT pg_terminate_backend(pid) FROM pg_stat_activity
WHERE username = 'some_role';
```

PostgreSQL では、データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。

deadlock_timeout

データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。

¹ データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。

データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。

データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。このコマンドは、データベースのバックグラウンドプロセスを強制終了する。

PostgreSQL 9.5 SELECT FOR UPDATE
SKIP LOCKED

statement_timeout

statement_timeout is a configuration parameter that specifies the maximum time in milliseconds that a SQL statement can run before it is terminated. The default value is 0, which means that there is no timeout. The parameter can be set to a non-zero value to prevent long-running queries from blocking other queries. The parameter is set in the postgresql.conf file and can be changed at runtime using the ALTER CONFIGURATION command.

lock_timeout

lock_timeout is a configuration parameter that specifies the maximum time in milliseconds that a query can wait for a lock before it is terminated. The default value is 0, which means that there is no timeout. The parameter can be set to a non-zero value to prevent long-running queries from blocking other queries. The parameter is set in the postgresql.conf file and can be changed at runtime using the ALTER CONFIGURATION command.

idle_in_transaction_session_timeout

idle_in_transaction_session_timeout is a configuration parameter that specifies the maximum time in milliseconds that a session can be idle in a transaction before it is terminated. The default value is 0, which means that there is no timeout. The parameter can be set to a non-zero value to prevent long-running transactions from blocking other queries. The parameter is set in the postgresql.conf file and can be changed at runtime using the ALTER CONFIGURATION command.

9.1 pg_stat_activity
9.2 procpid pid

PostgreSQL 9.6 pg_stat_activity
waiting true
wait_event_type wait_event
9.6 waiting
9.6 waiting = true

pgAdmin SQL 2-5 SQL

2-5

```
CREATE ROLE leo LOGIN PASSWORD 'king' VALID UNTIL 'infinity'  
CREATEDB;
```

[illegible]

2-6

2-6

```
CREATE ROLE regina LOGIN PASSWORD 'queen' VALID UNTIL '2020-1-1
00:00' SUPERUSER;
```

LOGIN
PASSWORD

2.3.2 ☐ ☐ ☐ ☐ ☐

[illegible]

SQL 入門

```
CREATE ROLE royalty INHERIT;
```

`inherit` の意味は、`royalty` が持つ権利を継承する。つまり、`PostgreSQL` は `INHERIT` を使って、`INHERIT` を使った。

NOINHERIT

```
GRANT royalty TO leo;  
GRANT royalty TO regina;
```

SUPERUSER SET
ROLE "SUPER" SUPERUSER

royalty

```
ALTER ROLE royalty SUPERUSER;
```

leo royalty leo
SUPERUSER SUPERUSER

```
SET ROLE royalty;
```

SUPERUSER

SUPERUSER

SET ROLE SET SESSION
AUTHORIZATION SUPERUSER
PostgreSQL current_user session_user

```
SELECT session_user, current_user;
```

SET ROLE current_user
SET SESSION AUTHORIZATION current_user

session_user

SET ROLE

- SET ROLE SUPERUSER
- SET ROLE current_user session_user
- SUPERUSER session_user SET ROLE
- SET ROLE session_user
- SET ROLE "SET SESSION AUTHORIZATION SET ROLE

SET SESSION AUTHORIZATION SET ROLE

- SET SESSION AUTHORIZATION
- SET SESSION AUTHORIZATION "SET SESSION AUTHORIZATION
- SET SESSION AUTHORIZATION current_user session_user
- session_user SET ROLE "SET SESSION AUTHORIZATION

SET ROLE SET SESSION AUTHORIZATION
leo 2-7

2-7 SET ROLE SET AUTHORIZATION

```
SELECT session_user, current_user;
```

```
session_user | current_user  
-----+-----  
leo          | leo  
(1 row)
```

```
SET SESSION AUTHORIZATION regina;
```

```
ERROR:  permission denied to set session authorization
```

```
SET ROLE regina;
```

```
ERROR:  permission denied to set role "regina"
```

```
ALTER ROLE leo SUPERUSER;
```

```
ERROR:  must be superuser to alter superusers
```

```
SET ROLE royalty;
```

```
SELECT session_user, current_user;
```

```
 session_user | current_user
-----+-----
leo           | royalty
(1 row)
```

```
SET ROLE regina;
```

```
ERROR:  permission denied to set role "regina"
```

```
ALTER ROLE leo SUPERUSER;
```

```
SET ROLE regina;
```

```
SELECT session_user, current_user;
```

```
 session_user | current_user
-----+-----
leo           | regina
(1 row)
```

```
SET SESSION AUTHORIZATION regina;
```

```
ERROR:  permission denied to set session authorization
```

```
--  leo
```

```
SELECT session_user, current_user;
```

```
SET SESSION AUTHORIZATION regina;
```

```
SELECT session_user, current_user;
```

```
 session_user | current_user
-----+-----
leo | leo
(1 row)
```

```
SET SESSION AUTHORIZATION
```

```
session_user | current_user
-----+-----
```

```
regina | regina
(1 row)
```

2-7 leo SET SESSION AUTHORIZATION SET ROLE regina regina leo SET ROLE royalty royalty leo royalty regina royalty SET ROLE leo royalty SET SESSION AUTHORIZATION regina session_user current_user regina

2.4 database

SQL database

```
CREATE DATABASE mydb;
```

template1 database CREATEDB database

2.4.1

database database PostgreSQL

PostgreSQL template0 template1 template1



template0 template1

template1 database template0 database

```
CREATE DATABASE my_db TEMPLATE my_template_db;
```

database database
PostgreSQL CREATEDB SQL

```
UPDATE pg_database SET datistemplate = TRUE WHERE datname = 'mydb';
```

datistemplate
FALSE

2.4.2 schema

schema database schema
public schema schema
schema IT
plane schema
employees schema
passengers schema

schema

SPA public schema
dogs

データベース

PostgreSQL をインストールして pgAdmin をインストールして
インストールしてインストールしてインストールしてインストールして pgAdmin をインストール
インストールしてインストールしてインストールして PostgreSQL をインストール
pgAdmin をインストールしてインストールしてインストールして 4.2.4 をインストール

2.5.1 インストール

PostgreSQL インストールしてインストールしてインストールして SELECT
INSERT UPDATE ALTER EXECUTE TRUNCATE

インストールしてインストールしてインストールしてインストールして ALTER
インストールしてインストールしてインストールして table1 ALTER
table2 SELECT function1 EXECUTE インストールして
インストールしてインストールしてインストールして EXECUTE インストールして

インストールしてインストールして CREATEDB CREATE ROLE インストール



PostgreSQL インストール privilege “” “”
right permission インストール

2.5.2 インストール

インストール PostgreSQL インストールしてインストールしてインストールしてインストールして
インストールして

(1) PostgreSQL インストールしてインストールしてインストールして database インストールして
postgres インストール postgres インストールして

(2) インストールして database インストールしてインストールして database インストールして
インストールしてインストールして

```
CREATE ROLE mydb_admin LOGIN PASSWORD 'something';
```

(3) database インストールして

```
CREATE DATABASE mydb WITH owner = mydb_admin;
```

(4) mydb_admin schema

2.5.3 GRANT

GRANT

```
GRANT some_privilege TO some_role;
```

GRANT

- GRANT
- DROP ALTER
- database schema
- WITH GRANT

```
GRANT ALL ON ALL TABLES IN SCHEMA public TO mydb_admin WITH  
GRANT OPTION;
```

- ALL ALL TABLES

```
GRANT SELECT, REFERENCES, TRIGGER ON  
ALL TABLES IN SCHEMA my_schema TO  
PUBLIC;
```

- PUBLIC

```
GRANT USAGE ON SCHEMA my_schema TO PUBLIC;
```

“GRANT” GRANT 2-8 PostgreSQL PUBLIC

PUBLIC CONNECT CREATE TEMP TABLE database EXECUTE REVOKE

```
REVOKE EXECUTE ON ALL FUNCTIONS IN SCHEMA my_schema FROM PUBLIC;
```

2.5.4



schema EXECUTE SELECT 2-8 PostgreSQL PUBLIC

2-8 schema

```
GRANT USAGE ON SCHEMA my_schema TO PUBLIC; ❶  
  
ALTER DEFAULT PRIVILEGES IN SCHEMA my_schema  
GRANT SELECT, REFERENCES ON TABLES TO PUBLIC; ❷  
  
ALTER DEFAULT PRIVILEGES IN SCHEMA my_schema  
GRANT ALL ON TABLES TO mydb_admin WITH GRANT OPTION; ❸  
  
ALTER DEFAULT PRIVILEGES IN SCHEMA my_schema ❹  
GRANT SELECT, UPDATE ON SEQUENCES TO public;  
  
ALTER DEFAULT PRIVILEGES IN SCHEMA my_schema ❺  
GRANT ALL ON FUNCTIONS TO mydb_admin WITH GRANT OPTION;
```

```
ALTER DEFAULT PRIVILEGES IN SCHEMA my_schema ⑥
GRANT USAGE ON TYPES TO PUBLIC;
```

❶ データベース database のスキーマ my_schema を取得する
データベース schema を取得するコマンドは schema の USAGE オプション
で指定した schema を取得する。スキーマ schema を取得する
コマンドは schema の USAGE オプションで指定する。

② `CREATE SCHEMA schema USAGE GUC; CREATE SCHEMA schema;
SELECT REFERENCE FROM INFORMATION_SCHEMA
TABLES;`

```
③ 创建 schema mydb_admin
mydb_admin 创建 schema
创建 schema mydb_admin
```

[illegible]

☐ " ☐ ☐ ☐ ☐ ☐ ☐ " ☐ ☐ ☐ ☐ ☐

2.5.5 PostgreSQL

[illegible]

```

##### PostgreSQL ##### database #####
#####
#####
#####
#####

```

```

schema [ ] schema [ ] USAGE [ ]

```

2.6 ☐ ☐ ☐ ☐ ☐

extension PostgreSQL contrib³ 9.1 extension PostgreSQL

3 extension contrib PostgreSQL 9.1 contrib extension



extension contrib PostgreSQL 9.1 contrib extension

PostgreSQL database PostgreSQL database 2.4.1 database

PostgreSQL PostgreSQL

database database 2-9

2-9 database

SELECT name, default_version, installed_version, left(comment,30) As comment FROM pg_available_extensions WHERE installed_version IS NOT NULL ORDER BY name;			
name	default_version	installed_version	comment
-----+-----+-----			
btree_gist	1.5	1.5	support for
indexing common da			
fuzzystrmatch	1.1	1.1	determine
similarities and dis			
hstore	1.4	1.4	data type
for storing sets of			
ogr_fdw	1.0	1.0	foreign-data
wrapper for GIS d			
pgrouting	2.4.1	2.4.1	pgRouting
Extension			

plpgsql	1.0	1.0	PL/pgSQL
procedural language			
plv8	1.4.10	1.4.10	
PL/JavaScript (v8) trusted pro			
postgis	2.4.0dev	2.4.0dev	PostGIS
geometry, geography, a			
(8 rows)			

postgresql database installed_version IS NOT NULL

postgresql database installed_version IS NOT NULL

```
\dx+ fuzzystmatch
```

postgresql database

```

SELECT pg_describe_object(D.classid,D.objid,0) AS description
FROM pg_catalog.pg_depend AS D INNER JOIN pg_catalog.pg_extension
AS E
ON D.refobjid = E.oid
WHERE
D.refclassid = 'pg_catalog.pg_extension'::pg_catalog.regclass AND
deptype = 'e' AND
E.extname = 'fuzzystmatch';

```

postgresql database

```

description
-----
function dmetaphone_alt(text)
function dmetaphone(text)
function difference(text,text)
function text_soundex(text)
function soundex(text)
function metaphone(text,integer)
function
levenshtein_less_equal(text,text,integer,integer,integer,integer)
function levenshtein_less_equal(text,text,integer)
function levenshtein(text,text,integer,integer,integer)

```

```
function levenshtein(text,text)
```

~~~~~  
~~~~~

2.6.1 安装

~~~~~  
database



~~~~~  
~~~~~

~~~~~ extension  
PostgreSQL contrib

01. 安装

~~~~~  
bin lib SQL  
share/extension 9.1 share/contrib  
9.1

~~~~~ PostgreSQL  
yum apt get postgresql-contrib
~~~~~  
PostGIS  
PostgreSQL  
~~~~~

```
SELECT * FROM pg_available_extensions;
```

02. database

PostgreSQL 9.1 以前は、CREATE EXTENSION ではなく、database ごとに、
pg_share/extension として、DROP EXTENSION が必要だった。
PostgreSQL 9.1 以降は、PostgreSQL Extension Network として、PostgreSQL
GitHub 上で、PostgreSQL “postgresql extension” として公開されている。

PostgreSQL fuzzystmatch 拡張機能

```
CREATE EXTENSION fuzzystmatch;
```

PostgreSQL psql から、database ごとに、
実行する。

```
psql -p 5432 -d mydb -c "CREATE EXTENSION fuzzystmatch;"
```



C 言語で、
C 言語

PostgreSQL schema ごとに、
schema ごとに

```
CREATE EXTENSION fuzzystmatch SCHEMA my_extensions;
```

03. PostgreSQL extension

PostgreSQL 9.1 以前は、9.1 以降は、contrib として、
extension として、PostgreSQL 9.1 以降は、
extension として、contrib として、
extension として、contrib として、

PostgreSQL 9.0 contrib schema tablefunc 9.1

```
CREATE EXTENSION tablefunc SCHEMA contrib FROM unpackaged;
```

contrib schema tablefunc extension pg_available_extensions extension contrib extension

contrib schema extension database extension

2.6.2

PostgreSQL PostgreSQL “” PostgreSQL

01.

9.1 PostgreSQL extension PL extension “” PostgreSQL PostgreSQL PostgreSQL

btree_gist

B-GiST B- 6.3.1

btree_gin

データベース B-ツリー GIN データベース B-ツリー
データベース B-ツリー 6.3.1

postgis

PostgreSQL データベース OGC GIS データベース 3D データ
データベース postgis データベース *PostGIS in Action* データ
PostGIS データベース PostGIS データベース “”
800 データベース PostGIS データベース
データベース PostGIS データベース
データベース pgpointcloud データベース
データベース pgRouting データベース

fuzzystmatch

データベース soundex
levenshtein データベース metaphone データベース “Where is Soundex
and Other Fuzzy Things” データベース

hstore

PostgreSQL データベース NoSQL データベース
hstore データベース hstore データベース jsonb データベース

pg_trgm **trigram**

データベース fuzzystmatch データベース
データベース ILIKE データベース LIKE
'%something%' データベース somefield ~
'(foo|bar) ' データベース “Teaching ILIKE
and LIKE New Tricks” データベース

dblink

2.7 备份工具

PostgreSQL 提供了很多备份工具，pg_dump 和 pg_dumpall 是 pg_basebackup 在 bin 目录下的

pg_dump 和 pg_dumpall 是 database 的 pg_dumpall 是 database 的 pg_dumpall 是 database 的 SUPERUSER 是 pg_basebackup 是 database 的

pg_dump 和 pg_dumpall 和 pg_basebackup 是 PostgreSQL 的 500GB 的 pg_basebackup 是 pg_basebackup 是 10.2 的

GNU 是 PostgreSQL 的 “”

PostgreSQL 的 pgBackRest 和 Barman 是

pgAgent 是 4.5 的 PostgreSQL 的 IP 的 localhost 的 localhsot 的

pg_dump 和 pg_dumpall 是 postgres 的 home 的 .pgpass 的 PGPASSWORD 的

2.7.1 pg_dump 备份数据库

pg_dump 和 pg_dumpall 是 PostgreSQL 数据库的备份工具。pg_dump 可以备份整个数据库或指定的 schema，而 pg_dumpall 可以备份整个数据库。pg_dump 支持多种输出格式，包括 SQL、TAR 和自定义格式。pg_restore 用于恢复备份。

pg_dump 的语法如下：
pg_dump [options] [dbname]

pg_dump 的选项包括：
-h host: 指定主机名。
-p port: 指定端口号。
-U user: 指定用户名。
-F format: 指定输出格式。
-b: 指定备份类型。
-v: 指定 verbosity。
-f filename: 指定输出文件名。

pg_dump 的数据库名是 mydb。

```
pg_dump -h localhost -p 5432 -U someuser -F c -b -v -f mydb.backup mydb
```

pg_dump 的数据库名是 mydb，输出格式是 SQL，备份类型是自定义格式，输出文件是 mydb.backup。

```
pg_dump -h localhost -p 5432 -U someuser -C -F p -b -v -f mydb.backup mydb
```

pg_dump 的数据库名是 mydb，输出格式是自定义格式，备份类型是自定义格式，输出文件是 mydb.backup。

```
pg_dump -h localhost -p 5432 -U someuser -F c -b -v -t *.pay* -f pay.backup mydb
```

pg_dump 的数据库名是 mydb，输出格式是自定义格式，备份类型是自定义格式，输出文件是 mydb.backup。

```
pg_dump -h localhost -p 5432 -U someuser -F c -b -v \n -n hr -n payroll -f hr.backup mydb
```

database public schema

```
pg_dump -h localhost -p 5432 -U someuser -F c -b -v -N public \
-f all_sch_except_pub.backup mydb
```

SQL INSERT PostgreSQL SQL PostgreSQL SQL SQL SQL

```
pg_dump -h localhost -p 5432 -U someuser -F p --column-inserts \
-f select_tables.backup mydb
```



"/path with spaces/mydb.backup" PostgreSQL

PostgreSQL 9.1 pg_dump 2-10 gzip

2-10

```
pg_dump -h localhost -p 5432 -U someuser -F d -f
/somepath/a_directory mydb
```

9.3 --jobs -j jobs=3 -j 3 2-11

2-11

```
pg_dump -h localhost -p 5432 -U someuser -j 3 -Fd -f
/somepath/a_directory mydb
```

2.7.2 pg_dumpall

pg_dumpall PostgreSQL database
SQL pg_dumpall SQL
B.2

```
pg_dumpall
database
pg_dump
database
pg_basebackup
PostgreSQL
pg_dumpall
SQL
SQL
pg_basebackup
PostgreSQL
```

□ □

```
pg_dumpall -h localhost -U postgres --port=5432 -f myglobals.sql --
globals-only
```

[illegible]

```
pg_dumpall -h localhost -U postgres --port=5432 -f myroles.sql --
roles-only
```

2.7.3 □□□□

PostgreSQL 数据库备份 pg_dump 与 pg_dumpall

- `psql` `pg_dump` `pg_dumpall` `SQL` `pg_dumpall`

- `pg_restore` 是 `pg_dump` 的逆操作，将 TAR 格式的数据还原。

- `psql` 是 SQL 客户端。

通过 SQL 客户端还原数据库。SQL 客户端还原数据库的步骤如下：
1. 使用 `psql` 客户端连接到数据库。
2. 使用 `psql` 客户端执行还原命令。

使用 SQL 客户端还原数据库的步骤如下：

```
psql -U postgres -f myglobals.sql
```

使用 SQL 客户端还原数据库的步骤如下：

```
psql -U postgres --set ON_ERROR_STOP=on -f myglobals.sql
```

使用 SQL 客户端还原数据库的步骤如下：
1. 使用 `psql` 客户端连接到数据库。

```
psql -U postgres -d mydb -f select_objects.sql
```

- `pg_restore` 是

用于还原数据库的工具。它可以将 `pg_dump` 生成的 TAR 格式的数据还原到数据库中。使用 `pg_restore` 还原数据库的步骤如下：

- 使用 `-j` 选项指定还原的并发数。例如：
`pg_restore -j 4`
- 使用 `pg_restore` 还原数据库。例如：
`pg_restore -d mydb`
- `pg_restore` 还原数据库时，可以使用 `database` 选项指定还原的数据库名称。例如：
`pg_restore -d mydb`

- pg_restore PostgreSQL のバックアップを PostgreSQL に戻す

pg_restore


database
database

9.2 使用 pg_restore --section 选项
pg_restore 选项用于指定要恢复的数据库部分。使用 --section 选项可以指定要恢复的数据库部分。使用 --section 选项可以指定要恢复的数据库部分。

```
CREATE DATABASE mydb2;
```

使用 pg_restore

```
pg_restore --dbname=mydb2 --section=pre-data --jobs=4  
mydb.backup
```

2.8 数据库配置

PostgreSQL 的“配置”选项用于配置数据库。

PostgreSQL 的默认配置位于 pg_default 文件中。pg_global 文件用于配置全局数据库。database 文件用于配置数据库。database 文件用于配置数据库。

2.8.1 配置

PostgreSQL 的默认配置位于 pg_default 文件中。pg_global 文件用于配置全局数据库。Windows 和 Unix 的默认配置位于 pg_default 文件中。

```
CREATE TABLESPACE secondary LOCATION 'C:/pgdata94_secondary';
```

Unix 的默认配置位于 fstab 文件中。

```
CREATE TABLESPACE secondary LOCATION  
'/usr/data/pgdata94_secondary';
```

2.8.2 数据库表空间

将数据库表空间移动到 secondary database 表空间

```
ALTER DATABASE mydb SET TABLESPACE secondary;
```

将表空间移动到 secondary

```
ALTER TABLE mytable SET TABLESPACE secondary;
```

PostgreSQL 9.4 将数据库表空间移动到 secondary database 表空间

将 pg_default 表空间移动到 secondary 表空间

```
ALTER TABLESPACE pg_default MOVE ALL TO secondary;
```

将数据库表空间移动到 secondary database 表空间

2.9 备份

使用 PostgreSQL 备份数据库

使用 pg_log 备份数据库 PostgreSQL 备份数据库

```
path/to/your/bin/pg_ctl -D your_postgresql_data_folder
```

2.9.1 备份 PostgreSQL 数据库

PostgreSQL 数据库的日志文件包括 pg_log、pg_xlog 和 pg_clog。pg_log 记录数据库的启动和关闭信息，pg_xlog 记录数据库的复制和恢复信息，pg_clog 记录数据库的 commit 和 rollback 信息。

pg_log 数据库的 data 目录。pg_xlog 数据库的 xlog 目录。pg_clog 数据库的 clog 目录。

pg_xlog 数据库的 xlog 目录。PostgreSQL 数据库的 log 目录。pg_clog 数据库的 clog 目录。

pg_xlog 数据库的 xlog 目录。pg_xlog 数据库的 xlog 目录。archive 数据库的 archive 目录。pg_xlog 数据库的 xlog 目录。PostgreSQL 数据库的 log 目录。pg_xlog 数据库的 xlog 目录。PostgreSQL 数据库的 log 目录。pg_xlog 数据库的 xlog 目录。PostgreSQL 数据库的 log 目录。pg_xlog 数据库的 xlog 目录。PostgreSQL 数据库的 log 目录。pg_xlog 数据库的 xlog 目录。PostgreSQL 数据库的 log 目录。

Windows 操作系统。PostgreSQL 数据库。Windows 操作系统。PostgreSQL 数据库。event viewer 事件查看器。



PostgreSQL 10 数据库的 pg_xlog 目录。pg_wal 目录。pg_clog 目录。pg_xact 目录。

2.9.2 PostgreSQL 数据库

postgres 数据库。PostgreSQL 数据库。postgres 数据库。PostgreSQL 数据库。

postgres 数据库。data 目录。PostgreSQL 数据库。postgres 数据库。

SQL 数据库系统安装与配置

数据库系统安装与配置 data 数据库系统安装与配置“第 1 章 / 第 1 章”数据库
第 1 章 postgres 数据库系统安装与配置数据库系统安装与配置 FDW 数据库
数据库系统安装与配置数据库系统安装与配置 postgres 数据库系统安装与配置
数据库系统安装与配置

2.9.3 配置 shared_buffers 数据库系统安装与配置

配置 shared_buffers 数据库系统安装与配置数据库系统安装与配置数据库系统安装与配置
数据库系统安装与配置 32 个 Windows 数据库系统安装与配置 512MB 数据库系统安装与配置
64 个 Windows 数据库系统安装与配置 1GB 数据库系统安装与配置 Linux 数据库系统安装与配置
配置 shared_buffers 数据库系统安装与配置 SHMMAX 数据库系统安装与配置 SHMMAX 数据库系统安装与配置
数据库系统安装与配置 shared_buffers 数据库系统安装与配置

PostgreSQL 9.3 数据库系统安装与配置数据库系统安装与配置数据库系统安装与配置数据库系统安装与配置
数据库系统安装与配置“数据库系统安装与配置”数据库系统安装与配置数据库系统安装与配置

2.9.4 配置 PostgreSQL 数据库系统安装与配置数据库系统安装与配置数据库系统安装与配置

配置 PostgreSQL 数据库系统安装与配置数据库系统安装与配置 pg_log 数据库系统安装与配置
make sure PostgreSQL is not already running 数据库系统安装与配置
数据库系统安装与配置

- postgres 数据库系统安装与配置数据库系统安装与配置数据库系统安装与配置
- PostgreSQL 数据库系统安装与配置数据库系统安装与配置
- postgres 数据库系统安装与配置数据库系统安装与配置 data 数据库系统安装与配置
postgresql.pid 数据库系统安装与配置数据库系统安装与配置
- 数据库系统安装与配置 PostgreSQL 数据库系统安装与配置数据库系统安装与配置数据库系统安装与配置数据库系统安装与配置
数据库系统安装与配置 PostgreSQL 数据库系统安装与配置数据库系统安装与配置

第 3 章 psql 数据库系统安装与配置

psql は PostgreSQL を操作するための SQL 言語を
実行するためのコマンドラインツールです。psql は
PostgreSQL のインストールパッケージに含まれており、
B.4 節で psql のインストール方法について説明します。

3.1 設定

PGHOST、PGPORT、PGUSER、PGPASSWORD の環境変数を psql に設定し、
PostgreSQL のデータベースに接続します。PostgreSQL 9.2
以降は psql が自動的に環境変数を取得するようになります。

PSQL_HISTORY

psql は、実行した SQL 文の履歴を ~/.psql_history に保存します。

PSQLRC

psql は、~/.psqlrc に保存された設定を読み込みます。

psql は、~/.psqlrc に保存された設定を読み込みます。



psql は、pgAdmin3 を利用してデータベースを操作できます。

3.2 psql のインストール

psql は、PostgreSQL のインストールパッケージに含まれており、
B.4 節で psql のインストール方法について説明します。

psql 是 PostgreSQL 的交互式 SQL 客户端。

psql 的用法在 \? 中给出。B.4 节给出了 psql 的 \h 命令。PostgreSQL 的文档中也有 psql 的用法。

psql 是 PostgreSQL 的交互式 SQL 客户端。psql 是 PostgreSQL 的交互式 SQL 客户端。psql 是 PostgreSQL 的交互式 SQL 客户端。pgAgent 是 Linux/Unix 上的定时任务工具。Windows 上的定时任务工具是 Task Scheduler。

psql 的用法在 B.5 节中给出。-f 选项用于指定要执行的脚本文件。

```
psql -f some_script_file
```

SQL 命令 -c 选项用于指定要执行的 SQL 命令。

```
psql -d postgresql_book -c "DROP TABLE IF EXISTS dress; CREATE SCHEMA staging;"
```

3-1 节中给出了 build_stage.psql 脚本的用法。staging.factfinder_import 脚本在 3-10 节中给出。CREATE TABLE 命令在 create_script.sql 脚本中给出。create_script.sql 脚本在 3-1 节中给出。

3-1 psql 的用法

```
\a ❶
\t
\g create_script.sql
SELECT
    'CREATE TABLE staging.factfinder_import (
        geo_id varchar(255), geo_id2 varchar(255), geo_display
```


PostgreSQL のインストールと設定
psql のインストールと設定 “psql のインストール”

図 3-2 psqlrc のインストールと設定 psql のインストール

図 3-2 psqlrc のインストール

```
\pset null 'NULL'
\encoding latin1
\set PROMPT1 '%n%M:%>%x %/# '
\pset pager always
\timing on
\set qstats92 '
    SELECT username, datname, left(query,100) || '...' As query
    FROM pg_stat_activity WHERE state != 'idle' ;
,
```



psqlrc の set のインストールと設定
psqlrc のインストールと設定

psql のインストールと設定

```
Null display is "NULL".
Timing is on.
Pager is always used.
psql (9.6beta3)
Type "help" for help.
postgres@localhost:5442 postgresql_book#
```

psql のインストール Linux/Unix のインストール Windows のインストール
psql のインストール Linux/Unix のインストール / のインストール
psql のインストール \ のインストール psql のインストール psqlrc のインストール
図 -X のインストール

psql のインストール psql のインストール psql のインストール
psql のインストール \unset のインストール \unset
qstat92 のインストール

psql を set して、データベースの接続情報を指定して起動します。
図 3-2 図 3-2 PROMPT1 を psql で起動する qstat92
PostgreSQL のインストール

3.3.1 psql の起動

psql は、データベースの接続情報を指定して起動します。
psql は、データベースの接続情報を指定して起動します。
database を指定して起動します。

```
\set PROMPT1 '%n%M:%>%x %/# '
```

psql は、データベースの接続情報を指定して起動します。
database を指定して起動します。
psql は、データベースの接続情報を指定して起動します。

psql は、データベースの接続情報を指定して起動します。

```
postgres@localhost:5442 postgresql_book#
```

\connect postgres_book を実行して、データベースに接続します。

```
postgres@localhost:5442 postgres_book#
```

3.3.2 テーブルの作成

psql は、データベースの接続情報を指定して起動します。
psql は、データベースの接続情報を指定して起動します。

psql は、データベースの接続情報を指定して起動します。
SELECT COUNT(*) FROM pg_tables を実行して、テーブルの数を取得します。

```
count
-----
73
```

```
(1 row)
Time: 18.650 ms
```

3.3.3 自動コミット

データベースのデフォルト動作として、SQL 実行時に自動的に DML 実行後にコミットが行われる。この動作をオフにすることで、明示的にコミットを行う必要がある。

データベースに接続後、\set AUTOCOMMIT off を実行して自動コミットをオフにする。

```
UPDATE census.facts SET short_name = 'This is a mistake.';
```

実行結果を確認する。

```
ROLLBACK;
```

実行結果を確認する。

```
COMMIT;
```



注意

psql では、\set AUTOCOMMIT off を実行すると、明示的に COMMIT を行う必要がある。

3.3.4 実行計画

psqlrc に \set eav 'EXPLAIN ANALYZE VERBOSE' を追加することで、SQL 実行時に実行計画と実行時間を見ることが出来る。

```
\set eav 'EXPLAIN ANALYZE VERBOSE'
```

EXPLAIN ANALYZE VERBOSE :eav
:

```
:eav SELECT COUNT(*) FROM pg_tables;
```

psqlrc 3-2

3.3.5

psql HISTSIZE
\set HISTSIZE 10
10

HISTFILE

```
\set HISTFILE ~/.psql_history - :DBNAME
```



Windows Cygwin/MingW
MSYS Unix

3.4 psql

psql

3.4.1 shell

psql \! Windows
psql \! dir

3.4.2 watch

\watch 在 PostgreSQL 9.3 中 psql 命令可以监视 SQL 语句的执行情况。
watch 命令可以监视 SQL 语句的执行情况 3-3 图

图 3-3 在 10 秒内监视 SQL 语句的执行情况

```
SELECT datname, query
FROM pg_stat_activity
WHERE state = 'active' AND pid != pg_backend_pid();
\watch 10
```

在 \watch 命令中，可以指定监视 SQL 语句的执行情况。在 \watch 命令中，可以指定监视 SQL 语句的执行情况。

图 3-4 在 5 秒内监视 SQL 语句的执行情况 5 秒内监视 SQL 语句的执行情况
\watch 命令可以监视 SQL 语句的执行情况 5 秒内

图 3-4 在 5 秒内监视 SQL 语句的执行情况

```
SELECT * INTO log_activity
FROM pg_stat_activity; ❶
INSERT INTO log_activity
SELECT * FROM pg_stat_activity; \watch 5 ❷
```

❶ 在 pg_stat_activity 表中插入数据到 log_activity 表

❷ 在 5 秒内监视 pg_stat_activity 表的执行情况 log_activity 表

在 watch 命令中，可以指定监视 SQL 语句的执行情况。在 watch 命令中，可以指定监视 SQL 语句的执行情况。CTRL-X 或 CTRL-C

3.4.3 监视 SQL 语句

在 psql 命令中，可以指定监视 SQL 语句的执行情况。在 psql 命令中，可以指定监视 SQL 语句的执行情况。3-5 图
pg_catalog 表 pg_t 监视 SQL 语句的执行情况

图 3-5 在 \dt+ 命令中监视 SQL 语句的执行情况

```
\dt+ pg_catalog.pg_t*
```

| Schema | Name | Type | Owner | Size | Description |
|------------|------------------|--------|----------|--------|-------------|
| -----+ | -----+ | -----+ | -----+ | -----+ | ----- |
| pg_catalog | pg_tablespace | table | postgres | 40 kB | |
| pg_catalog | pg_trigger | table | postgres | 16 kB | |
| pg_catalog | pg_ts_config | table | postgres | 40 kB | |
| pg_catalog | pg_ts_config_map | table | postgres | 48 kB | |
| pg_catalog | pg_ts_dict | table | postgres | 40 kB | |
| pg_catalog | pg_ts_parser | table | postgres | 40 kB | |
| pg_catalog | pg_ts_template | table | postgres | 40 kB | |
| pg_catalog | pg_type | table | postgres | 112 kB | |

データベースのテーブルを列挙する \d+ コマンド 3-6

3-6 \d+ コマンドの出力結果

```
\d+ pg_ts_dict
```

Table "pg_catalog.pg_ts_dict"

| Column | Type | Modifiers | Storage | Stats target |
|----------------|--------|-----------|----------|--------------|
| -----+ | -----+ | -----+ | -----+ | -----+ |
| dictname | name | not null | plain | |
| dictnamespace | oid | not null | plain | |
| dictowner | oid | not null | plain | |
| dicttemplate | oid | not null | plain | |
| dictinitoption | text | | extended | |

Indexes:

"pg_ts_dict_dictname_index" UNIQUE, btree (dictname, dictnamespace)

"pg_ts_dict_oid_index" UNIQUE, btree (oid)

Has OIDs: yes

3.4.4 テーブル

PostgreSQL 9.6 の psql コマンド \crosstabview を利用して、
データベースのテーブルを列挙する psql コマンド 3-7
データベースのテーブルを列挙する

例 3-7 表関数

```
SELECT student, subject, AVG(score)::numeric(5,2) As avg_score
FROM test_scores
GROUP BY student, subject
ORDER BY student, subject
\crosstabview student subject avg_score
```

| student | algebra | calculus | chemistry | physics | scheme |
|---------|---------|----------|-----------|---------|--------|
| alex | 74.00 | 73.50 | 82.00 | 81.00 | |
| leo | 82.00 | 65.50 | 75.50 | 72.00 | |
| regina | 72.50 | 64.50 | 73.50 | 84.00 | 90.00 |
| sonia | 76.50 | 67.50 | 84.00 | 72.00 | |

(4 rows)

\crosstabview 関数は、SQL 関数 \crosstabview
 を呼び出す。SQL 関数は、データベースに格納されたデータを
 表形式で表示する。 \crosstabview 関数は、SQL
 関数 \crosstabview を呼び出す。

例 3-7 student 表の subject 列の平均値
 avg_score を表示する。 student-subject 表を \crosstabview
 関数で呼び出す。 SQL 関数は、データベースに格納されたデータを

3.4.5 SQL

データベースに格納されたデータを SQL 関数 PostgreSQL 9.6
 関数 SQL 関数 PostgreSQL 9.6 関数 SQL 関数 PostgreSQL 9.6
 \gexec 関数 SQL 関数 SQL 関数 SQL 関数 SQL 関数 SQL
 関数 SQL 関数 SQL 関数 SQL 関数 SQL 関数 SQL 関数 SQL
 関数 gexec 関数 SQL 関数 SQL 関数 SQL 関数 SQL 関数 SQL
 SQL 関数 gexec 関数 SQL 関数 SQL 関数 SQL 関数 SQL
 関数 SQL 関数 gexec 関数 SQL 関数 SQL 関数 SQL 関数 SQL
 null 関数 3-8 関数 \gexec 関数

例 3-8 gexec の実行

```
SELECT
  'CREATE TABLE ' || person.name || '( a integer, b integer)' As
create,
  'INSERT INTO ' || person.name || ' VALUES(1,2) ' AS insert
FROM (VALUES ('leo'),('regina')) AS person (name) \gexec

CREATE TABLE
INSERT 0 1
CREATE TABLE
INSERT 0 1
```

例 3-9 gexec の実行 information_schema の検索

例 3-9 gexec の実行

```
SELECT
  'SELECT ' || quote_literal(table_name) || ' AS table_name,
COUNT(*) As count FROM ' || quote_ident(table_name) AS cnt_q
FROM information_schema.tables
WHERE table_name IN ('leo','regina') \gexec

table_name | count
-----+-----
leo        | 1
(1 row)

table_name | count
-----+-----
regina     | 1
(1 row)
```

3.5 psql のインストール

psql をインストール \copy のインストール

psql のインストール

psql のインストール

psql のインストール


```
\copy sometable FROM somefile.txt NULL As '';
```



psql \copy SQL COPY
psql SQL copy
postgres postgres

3.5.2 psql

psql \copy 3-11

3-11 psql

```
\connect postgresql_book
\copy (SELECT * FROM staging.factfinder_import WHERE s01 ~ E'^[0-9]+' )
TO '/test.tab'
WITH DELIMITER E'\t' CSV HEADER
```

psql tab
HEADER CSV 3-12

3-12 psql

```
\connect postgresql_book
\copy staging.factfinder_import TO '/test.csv'
WITH CSV HEADER QUOTE '"' FORCE QUOTE *
```

FORCE QUOTE *
FORCE QUOTE *

3.5.3

3-13 psql 使用說明

Hubert Lubaczewski “Piping copy to/from an external program”

psql HTML
HTML 3-1

| category | num_per_cat |
|---|-------------|
| Query Tuning / Genetic Query Optimizer | 7 |
| Query Tuning / Other Planner Options | 5 |
| Query Tuning / Planner Cost Constants | 6 |
| Query Tuning / Planner Method Configuration | 11 |
| Statistics / Query and Index Statistics Collector | 6 |

3-1 HTML

⑤ 生成HTML 文件

⑥ string_agg() PostgreSQL 9.0 字符串聚合函数

⑦ 数据库性能调优之数据库配置参数

⑧ “数据库”数据库配置参数

3-14 生成HTML 文件 SQL 命令 psql 生成HTML 文件
3-14 生成HTML 文件 psql 生成HTML 文件 \i
settings_report.psql 生成HTML 文件 psql -f
settings_report.psql 生成HTML 文件 settings_report.html 生成HTML 文件 3-2
生成HTML 文件

Planner Settings

| | |
|----------|--|
| category | Query Tuning / Other Planner Options |
| settings | constraint_exclusion=partition
cursor_tuple_fraction=0.1
default_statistics_target=100
from_collapse_limit=8
join_collapse_limit=8 |
| category | Query Tuning / Planner Cost Constants |
| settings | cpu_index_tuple_cost=0.005
cpu_operator_cost=0.0025
cpu_tuple_cost=0.01
effective_cache_size=16384
random_page_cost=4
seq_page_cost=1 |
| category | Query Tuning / Planner Method Configuration |
| settings | enable_bitmapscan=on
enable_hashagg=on
enable_hashjoin=on
enable_indexonlyscan=on
enable_indexscan=on
enable_material=on |

File Locations

| | |
|-------------------|---|
| config_file | C:/projects/pg/pg92edb/data/postgresql.conf |
| data_directory | C:/projects/pg/pg92edb/data |
| external_pid_file | |
| hba_file | C:/projects/pg/pg92edb/data/pg_hba.conf |
| ident_file | C:/projects/pg/pg92edb/data/pg_ident.conf |

Memory Settings

| | | |
|---------------------------|-------|----|
| maintenance_work_mem | 16384 | kB |
| max_prepared_transactions | 0 | |
| max_stack_depth | 2048 | kB |
| shared_buffers | 4096 | kB |
| temp_buffers | 1024 | kB |
| track_activity_query_size | 1024 | |
| work_mem | 1024 | kB |

3-2 生成HTML 文件

生成HTML 文件 psql 生成HTML 文件
pgAgent cronab Windows 生成HTML 文件

4 pgAdmin

pgAdmin4 是 PostgreSQL 数据库管理工具。V1.6 版本支持 pgAdmin3 和 pgAdmin4。pgAdmin3 是旧版本，pgAdmin4 是最新版本。pgAdmin4 支持 PostgreSQL 9.6 和 PostgreSQL 10。pgAdmin3 支持 PostgreSQL 9.5 和 PostgreSQL 9.6。pgAdmin4 支持 PostgreSQL 9.6 和 PostgreSQL 10。pgAdmin3 支持 PostgreSQL 9.5 和 PostgreSQL 9.6。pgAdmin4 支持 PostgreSQL 9.6 和 PostgreSQL 10。pgAdmin3 支持 PostgreSQL 9.5 和 PostgreSQL 9.6。pgAdmin4 支持 PostgreSQL 9.6 和 PostgreSQL 10。



pgAdmin4 是 PostgreSQL 9.6 和 PostgreSQL 10 的数据库管理工具。pgAdmin3 是 PostgreSQL 9.5 和 PostgreSQL 9.6 的数据库管理工具。pgAdmin4 支持 PostgreSQL 9.6 和 PostgreSQL 10。pgAdmin3 支持 PostgreSQL 9.5 和 PostgreSQL 9.6。pgAdmin4 支持 PostgreSQL 9.6 和 PostgreSQL 10。pgAdmin3 支持 PostgreSQL 9.5 和 PostgreSQL 9.6。pgAdmin4 支持 PostgreSQL 9.6 和 PostgreSQL 10。

pgAdmin 是 PostgreSQL 数据库管理工具。pgAdmin4 是 PostgreSQL 9.6 和 PostgreSQL 10 的数据库管理工具。pgAdmin3 是 PostgreSQL 9.5 和 PostgreSQL 9.6 的数据库管理工具。pgAdmin4 支持 PostgreSQL 9.6 和 PostgreSQL 10。pgAdmin3 支持 PostgreSQL 9.5 和 PostgreSQL 9.6。pgAdmin4 支持 PostgreSQL 9.6 和 PostgreSQL 10。pgAdmin3 支持 PostgreSQL 9.5 和 PostgreSQL 9.6。pgAdmin4 支持 PostgreSQL 9.6 和 PostgreSQL 10。

4.1 pgAdmin

PostgreSQL 数据库管理工具 pgAdmin4 和 BigSQL 是 EDB 的 PostgreSQL 9.6 版本。pgAdmin4 是 PostgreSQL 9.6 版本。pgAdmin3 是 PostgreSQL 9.6 版本。BigSQL 是 PostgreSQL 9.6 版本。pgAdmin3 LTS 是 PostgreSQL 9.6 版本。PostgreSQL 10 版本。BigSQL 是 PostgreSQL 9.6 版本。pgAdmin3 LTS 是 PostgreSQL 9.5 版本。EDB 是 PostgreSQL 9.5 版本。pgAdmin4 是 PostgreSQL 9.6 版本。pgAdmin3 是 PostgreSQL 9.6 版本。

pgAdmin PostgreSQL pgAdmin
pgAdmin PostgreSQL
“”PostgreSQL
bug

4.1.1 〇〇〇〇

```

pgAdmin

```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □

□ □ □ □ □ □ □ □ □ □

[illegible]

SQL 練習問題

pgAdmin SQL PostgreSQL
SQL
pgAdmin SQL SQL SQL SQL
SQL SQL SQL

postgresql.conf □ **pg_hba.conf** □□□□□□□□□□

```

pgAdmin
pgAdmin3
postgres database
pgadmin

```

□ □ □ □ □ □ □

pgAdmin CSV pgAdmin3 HTML

pg_restore pg_dump

pgAdmin

pg_restore pg_dump pgAdmin database schema " " pg_dump pg_restore

pgAdmin

pgAdmin

pgScript

pgScript " " pgScript pgAdmin3 pgAdmin4

SQL

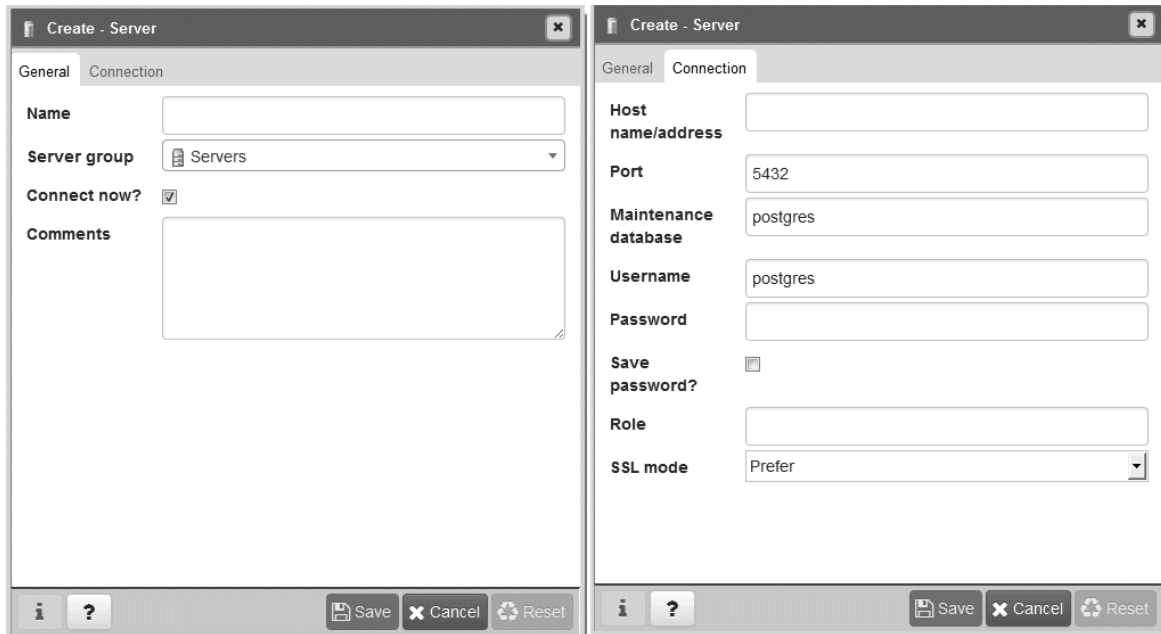
CTRL pgAdmin4

pgAgent

pgAgent pgAdmin pgAgent

4.1.2 PostgreSQL

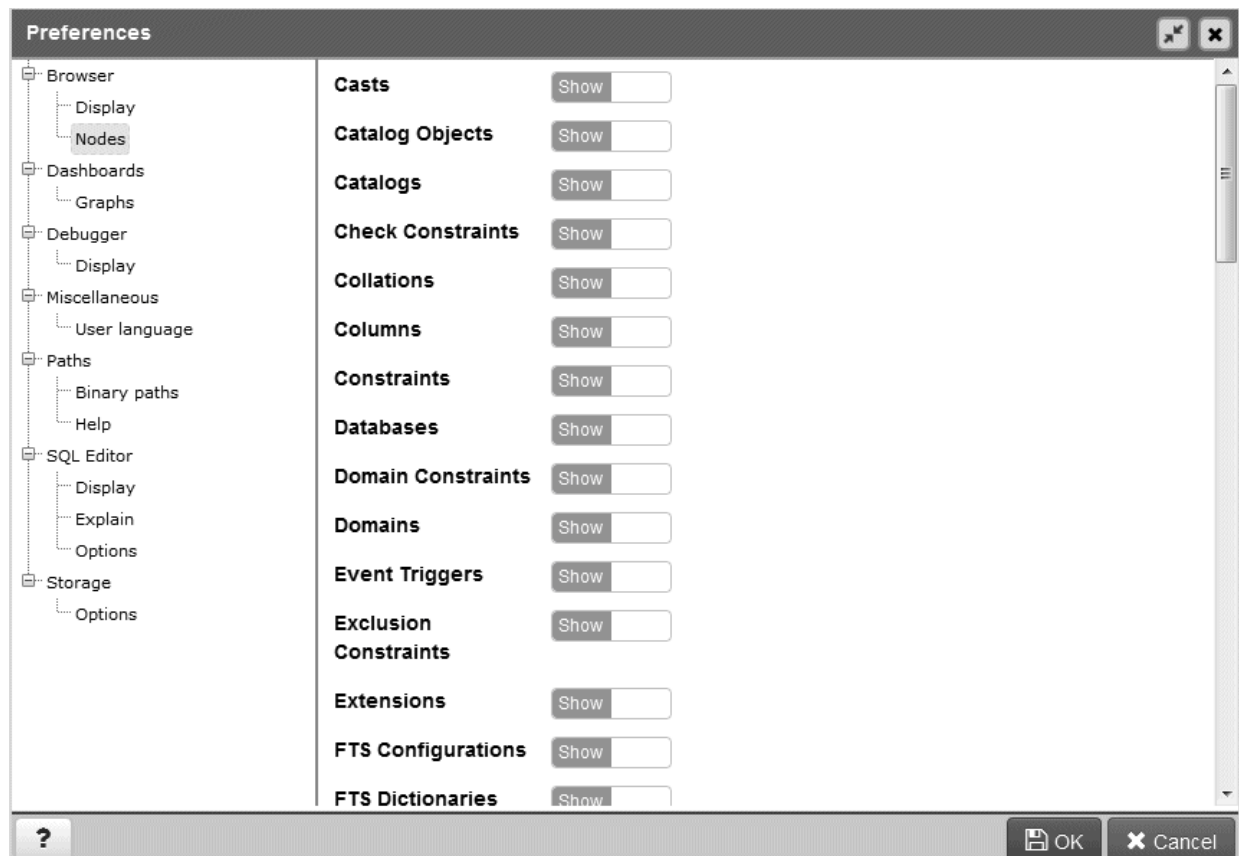
pgAdmin PostgreSQL General Connection 4-1



■ 4-1 pgAdmin4 のインストール

4.1.3 pgAdmin のインストール

pgAdmin のインストールは、以下の手順で行います。
 ① ② ③ Files → Preferences のインストール Nodes のインストール
 ④ ⑤ ⑥ Files → Preferences のインストール Browser のインストール Nodes のインストール
 ⑦ ⑧ ⑨ 4-2 のインストール



4-2 pgAdmin4 のインストールと起動

pgAdmin 4 のインストールは、以下の手順で行います。

1. Browser → Display のメニューから「pgAdmin 4」を選択します。

2. PostgreSQL のインストール時に「pgAdmin 4」を選択します。

3. PostgreSQL の schema として「information_schema」を選択します。

4. pg_catalog として「catalog」を選択します。

5. ANSI SQL として「MySQL」を選択します。

6. SQL Server として「SQL Server」を選択します。

4.2 pgAdmin のインストール

pgAdmin 4 のインストールは、以下の手順で行います。

4.2.1 pgAdmin 4 のインストール

pgAdmin 4 のインストールは、以下の手順で行います。

1. pgAdmin 4 のインストールスクリプトをダウンロードします。

2. Scripts フォルダにインストールスクリプトを配置します。

3. 4-3 の手順に従ってインストールを行います。

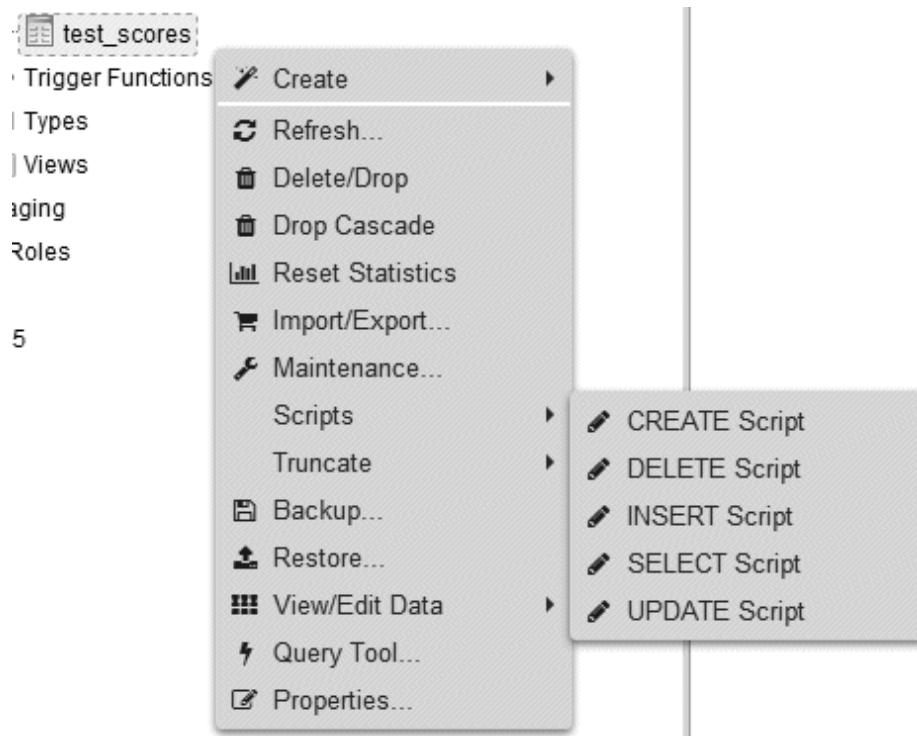


図 4-3 右クリックメニュー

図 4-3 “SELECT Script”メニューを選択すると、図 4-4 のように PSQL Console が起動します。この画面で SQL を実行できます。

4.2.2 pgAdmin3 から psql

pgAdmin から psql を実行するには、pgAdmin3 のメニューから psql を実行します。pgAdmin3 のメニューから psql を実行すると、図 4-4 のように PSQL Console が起動します。この画面で SQL を実行できます。



図 4-4 psql の起動

データベースをインストール database をインストール pgAdmin をインストール PostgreSQL をインストール database をインストール “PSQL Console” をインストール

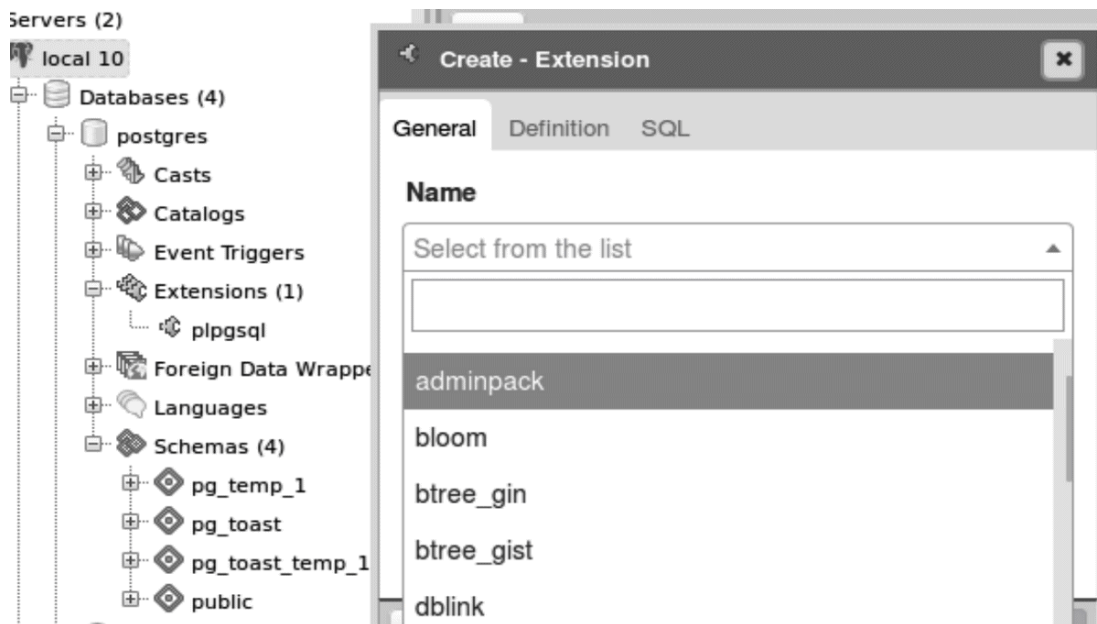
4.2.3 pgAdmin3 postgresql.conf pg_hba.conf

adminpack をインストール pgAdmin をインストール PostgreSQL をインストール adminpack をインストール Server Configuration をインストール 4-5



4-5 pgAdmin3

pgAdmin PostgreSQL Server Configuration adminpack adminpack CREATE EXTENSION adminpack 4-6



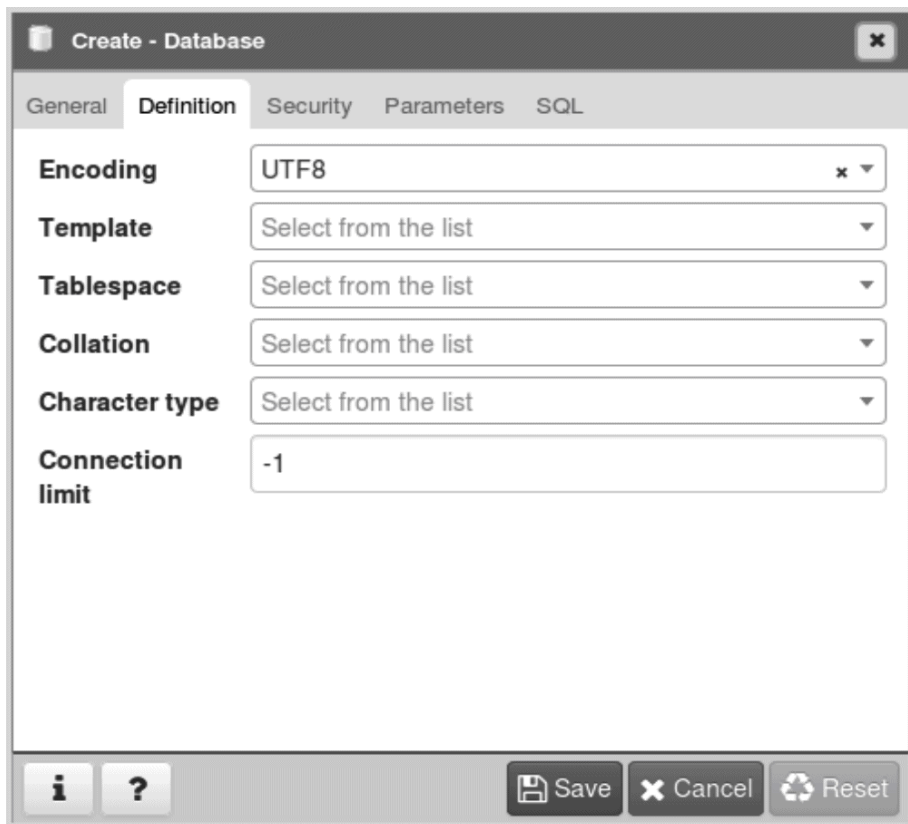
■ 4-6 pgAdmin4 のインストール

4.2.4 インストール手順

pgAdmin インストール手順

01. インストール

pgAdmin インストール手順 database 作成
New Database 4-7 Definition
2.4.1 インストール

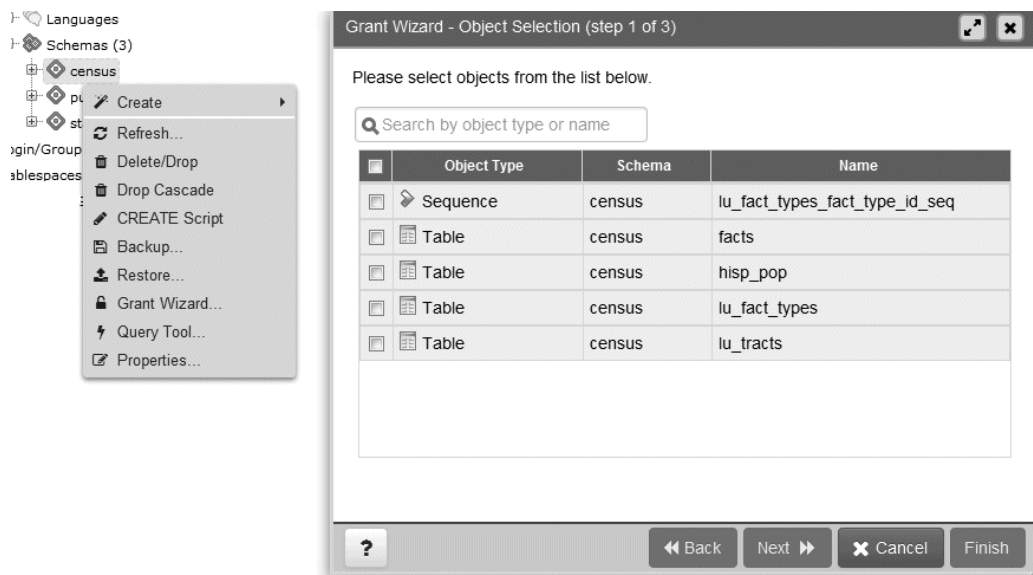


■ 4-7 pgAdmin4 のインストール

schema インストール

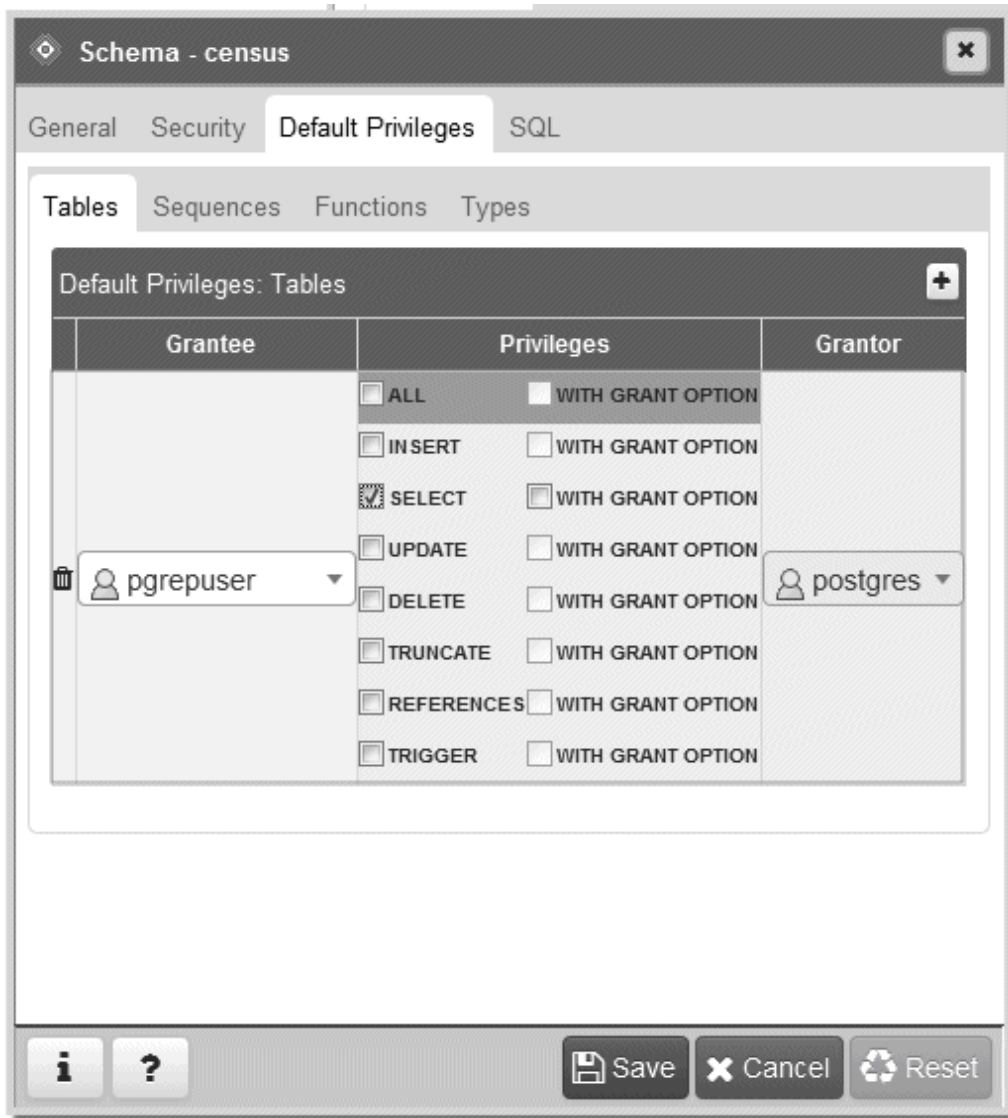
02. インストール

PostgreSQL をインストールした後は pgAdmin をインストールして
 ツールメニュー → Grant Wizard を起動して schema を選択して
 Grant Wizard を起動して schema を選択して
 Privileges を設定する



4-8 pgAdmin4 のインストール

PostgreSQL をインストールした後は pgAdmin をインストールして
 schema を database
 schema を database
 Properties を設定して Default
 Privileges を設定する



4-9 pgAdmin4 のインストール

この schema のインストールは、pgAdmin4 のインストールと同様である。

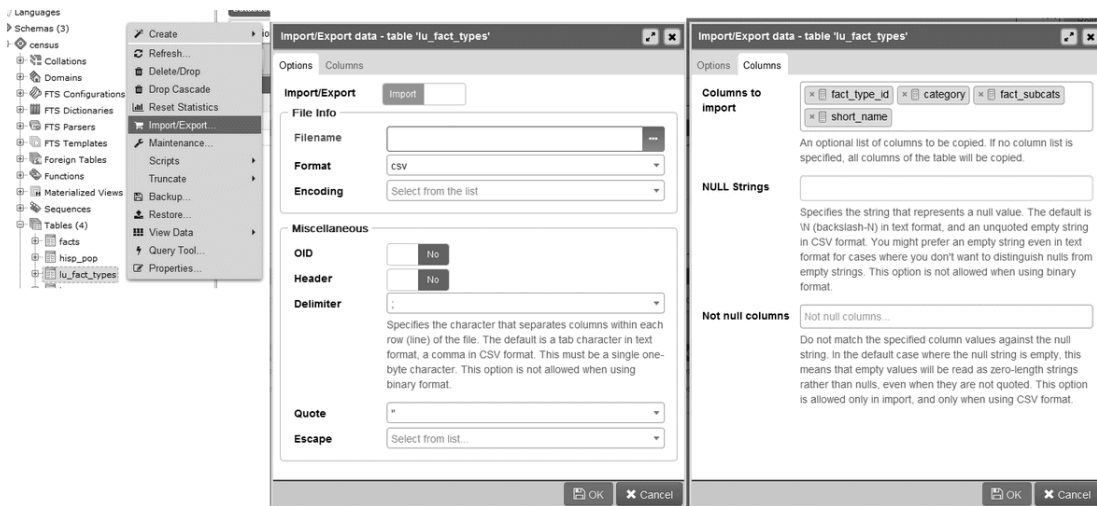
4.2.5 インストール

psql を pgAdmin からインストールする。

01. インストール

pgAdmin から psql をインストールする。 \copy コマンドを使用して、4-10 のインストール

lu_fact_types



4-10 pgAdmin4 Import

02. pgAdmin

pgAdmin3 CSV HTML XML pgAdmin4 pgAdmin3 **

pgAdmin

(1) Query Tool

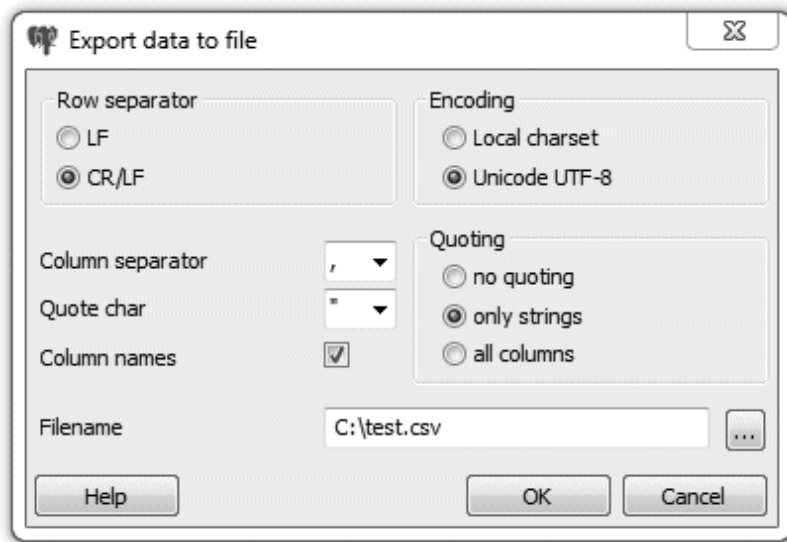
(2)

(3)

(4) pgAdmin3 File → Export

pgAdmin4

(5) pgAdmin3 pgAdmin4 4-11



4-11 Export

HTML XML File →
Quick Report 4-12

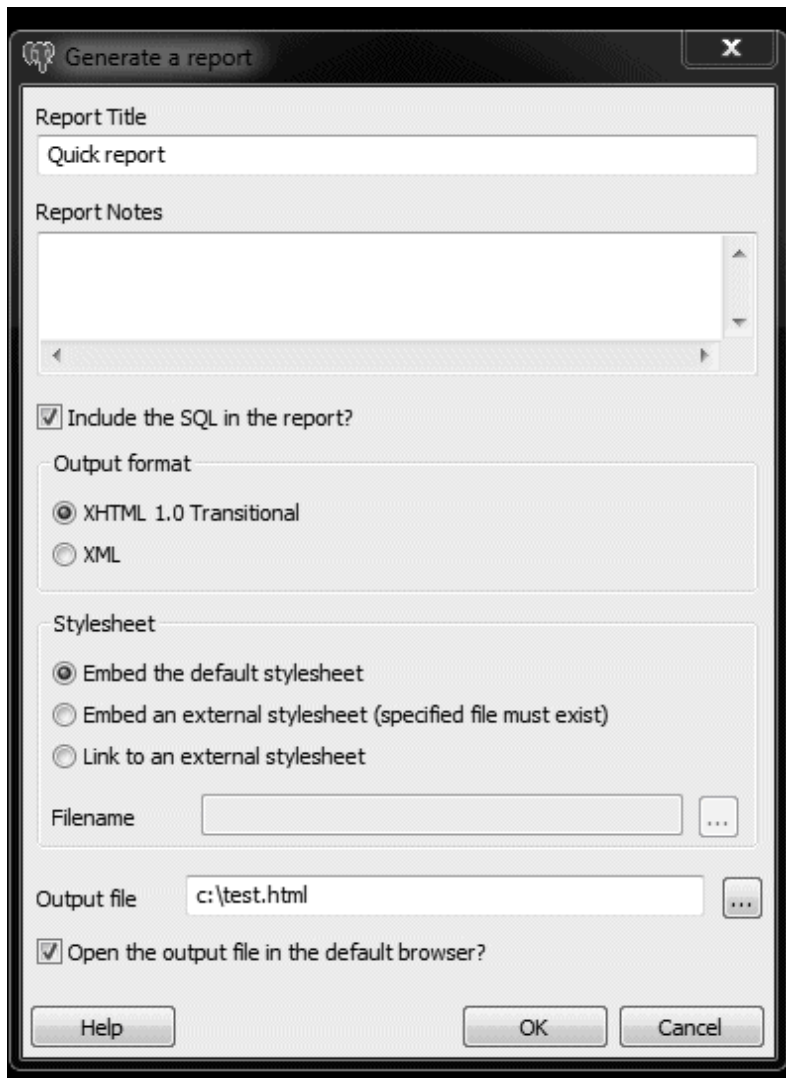
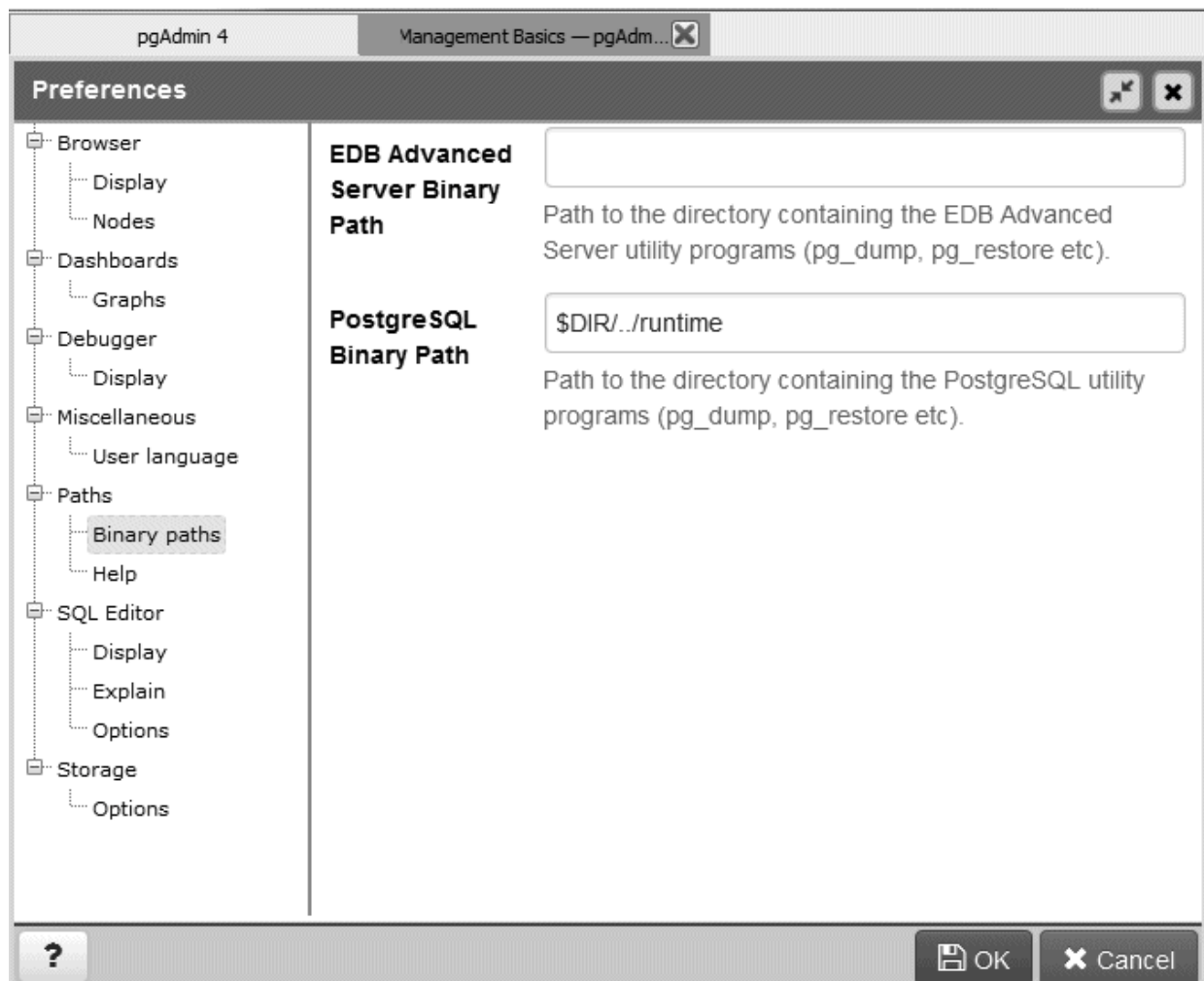


図 4-12 レポートの生成

4.2.6 データのバックアップ

pgAdmin は pg_dump と pg_restore を利用してデータベースのバックアップと復元を行う。2.7 節で説明したように、pgAdmin は PostgreSQL の bin ディレクトリに pg_dump と pg_restore が含まれている。

PostgreSQL の bin ディレクトリに pgAdmin がインストールされている場合、pgAdmin は PostgreSQL の bin ディレクトリに pg_dump と pg_restore が含まれていることを検出する。pgAdmin の bin ディレクトリに pg_dump と pg_restore が含まれていない場合は、図 4-13 のようにエラーメッセージが表示される。



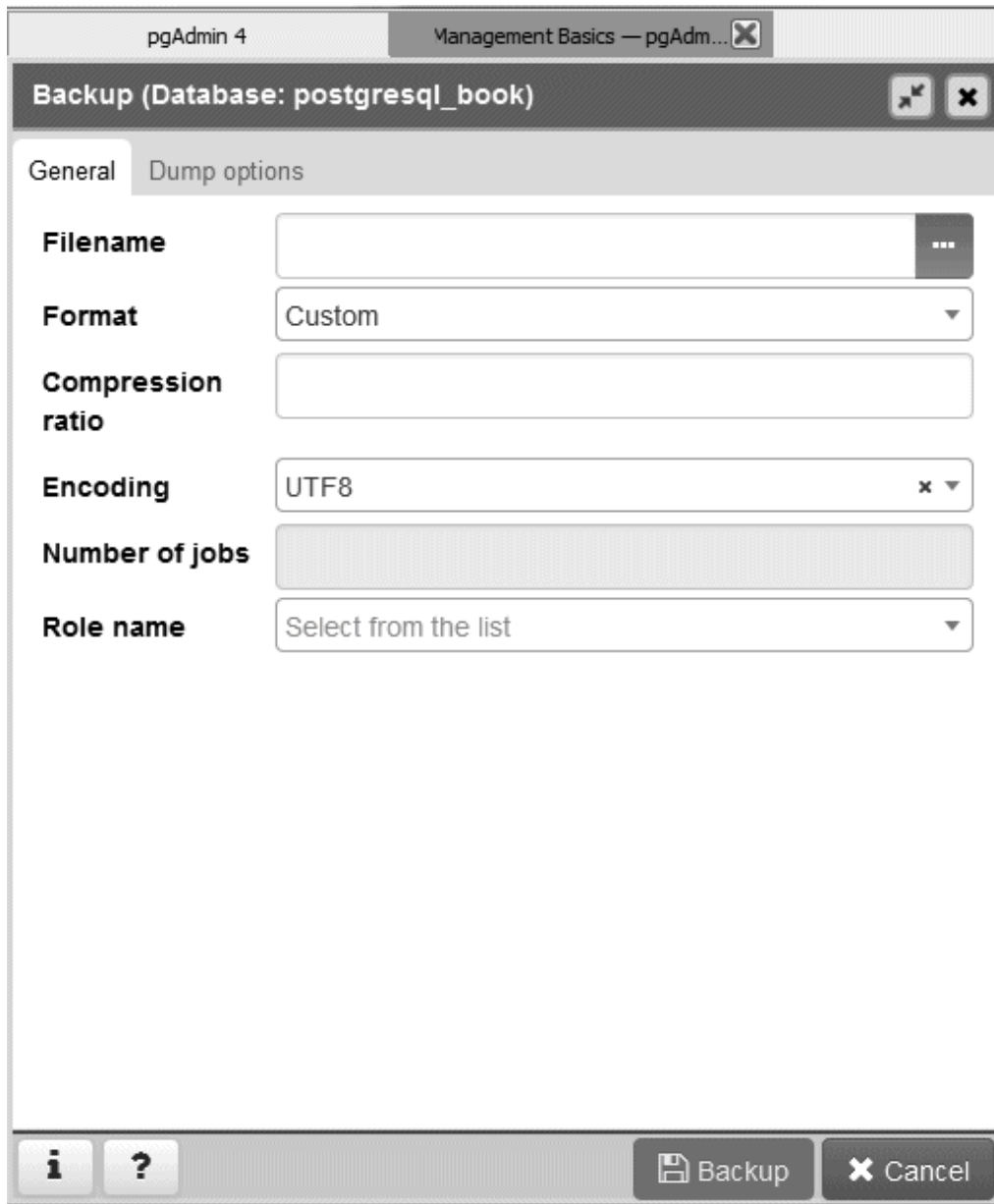
4-13 pgAdmin File → Preferences



pgAdminは、PostgreSQLの管理を行うためのツールです。pgAdminのインストールディレクトリにpg_dumpやpg_restoreなどのユーティリティプログラムが含まれています。pgAdminのインストールディレクトリにpg_dumpやpg_restoreなどのユーティリティプログラムが含まれています。

01. database

2.7.1 database pgAdmin database Custom 4-14



4-14 database

02.

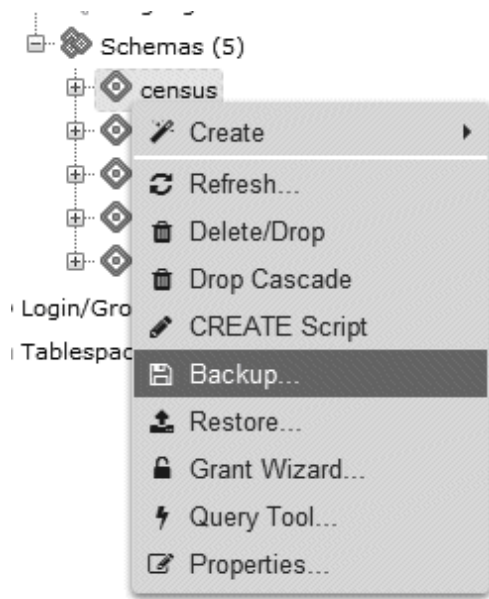
pgAdmin pg_dumpall PostgreSQL Tools → Backup Globals

pgAdmin pg_dumpall
pgAdmin

Tools → Backup
Server

03. pgAdmin

pgAdmin pg_dump
Backup 4-15
database schema



4-15 pgAdmin schema

pgAdmin3
4-14 Objects
4-16 pgAdmin4

pgScript 是一个用于在 SQL Server 中执行 Transact-SQL 脚本的工具。它可以在 SQL Server 的 DECLARE SET IF-ELSE WHILE 等语句中使用 pgScript 来执行。

在 SQL 脚本中使用 pgScript 时，需要在 pgScript 之前添加一个  图标。

在 pgScript 脚本中，4-1 是一个用于 pgScript 的脚本。它使用 lu_fact_types 表中的 7-22 数据。pgScript 脚本中 census.hisp_pop 表中的 hispanic_or_latino 和 white_alone 以及 black_or_african_american_alone 等列。

图 4-1 在 pgScript 中使用脚本

```
DECLARE @I, @labels, @tdef;
SET @I = 0;

--labels
SET @labels =
    SELECT
        quote_ident(
            replace(
                replace(lower(COALESCE(fact_subcats[4],
fact_subcats[3])), ' ', '_'),':','')
            ) As col_name,
        fact_type_id
    FROM census.lu_fact_types
    WHERE category = 'Population' AND fact_subcats[3] ILIKE
'Hispanic or Latino%'
    ORDER BY short_name;

SET @tdef = 'census.hisp_pop(tract_id varchar(11) PRIMARY KEY ';

--LINES
WHILE @I < LINES(@labels)
BEGIN
    SET @tdef = @tdef + ', ' + @labels[@I][0] + ' numeric(12,3) ';
    SET @I = @I + 1;
END
```

```
SET @tdef = @tdef + ')';
```

```
GO
PRINT @tdef;
```

```
GO
CREATE TABLE @tdef;
```

pgScript 4-1 SQL 4-2 pgScript census.hisp_pop

4-2 pgScript

```
DECLARE @I, @labels, @tload, @tcols, @fact_types;
SET @I = 0;
SET @labels =
    SELECT
        quote_ident(
            replace(
                replace(
                    lower(COALESCE(fact_subcats[4],
fact_subcats[3])), ' ', '_'), ':', ''
                ) As col_name,
        fact_type_id
    FROM census.lu_fact_types
    WHERE category = 'Population' AND fact_subcats[3] ILIKE
'Hispanic or Latino%'
    ORDER BY short_name;

SET @tload = 'tract_id';
SET @tcols = 'tract_id';
SET @fact_types = '-1';

WHILE @I < LINES(@labels)
BEGIN
    SET @tcols = @tcols + ', ' + @labels[@I][0] ;
    SET @tload = @tload +
        ', MAX(CASE WHEN fact_type_id= ' +
        CAST(@labels[@I][1] AS STRING) +
        ' THEN val ELSE NULL END)';
    SET @fact_types = @fact_types + ', ' + CAST(@labels[@I][1] As
STRING);
    SET @I = @I + 1;
```




census.hisp_pop



Sort



Aggregate

```
Temp Written Blocks 0
Sort Key             ("left"((hisp_pop.tract_id)::text, 5))
Node Type            Sort
Sort Space Used      129
Actual Total Time    0.938
Shared Hit Blocks    15
Shared Read Blocks   0
Local Hit Blocks     0
Local Dirtied Blocks 0
Sort Method          quicksort
Plan Width           40
Actual Loops         1
Actual Startup Time  0.883
Temp Read Blocks     0
Output               ("left"((tract_id)::text, 5)),hispanic_or_latino,white_alone
Local Read Blocks    0
Sort Space Type      Memory
Startup Cost         111.29
Shared Dirtied Blocks 0
Shared Written Blocks 0
Local Written Blocks 0
Plan Rows            1478
Parallel Aware       false
Actual Rows          1478
Parent Relationship   Outer
Total Cost           114.98
```

4-17

SQL Query → Explain → Buffers
Data Output

```
GroupAggregate (cost=111.29..151.93 rows=1478 width=20)
  Output: ("left"((tract_id)::text, 5)), sum(hispanic_or_latino),
  sum(white_alone), ...
  -> Sort (cost=111.29..114.98 rows=1478 width=20)
    Output: tract_id, hispanic_or_latino, white_alone,
    ("left"((tract_id)::text, 5))
    Sort Key: ("left"((tract_id)::text, 5))
    -> Seq Scan on census.hisp_pop (cost=0.00..33.48 rows=1478
width=20)
      Output: tract_id, hispanic_or_latino
```


■ 4-18 pgAgent と pgAdmin4

pgAgent と pgAdmin4 は PostgreSQL の管理ツールです。pgAgent は SQL を定期的に実行するためのスケジューリングツールで、pgAdmin4 は PostgreSQL の管理ツールです。pgAgent は pgAdmin4 から設定できます。



pgAgent は PostgreSQL のスケジューリングツールです。pgAgent は pgAdmin4 から設定できます。pgAgent は PostgreSQL のスケジューリングツールです。pgAgent は pgAdmin4 から設定できます。

4.5.2 pgAgent

pgAgent は PostgreSQL のスケジューリングツールです。pgAgent は pgAdmin4 から設定できます。pgAgent は PostgreSQL のスケジューリングツールです。pgAgent は pgAdmin4 から設定できます。

The screenshot shows the 'Create - pgAgent Job' dialog box with the 'General' tab selected. The job name is 'Vacuum tables'. The 'Enabled?' checkbox is checked. The 'Kind' is set to 'SQL'. The 'Connection type' is set to 'Local'. The 'Database' is set to 'postgres_book'. The 'On error' action is set to 'Fail'. The 'Comment' field is empty.

The screenshot shows the 'Create - pgAgent Job' dialog box with the 'SQL' tab selected. The 'SQL query' field contains the text '1 vacuum analyze verbose;'. The 'On error' action is set to 'Fail'.

■ 4-19 pgAdmin

pgAdmin は PostgreSQL の管理ツールです。pgAdmin は SQL を定期的に実行するためのスケジューリングツールで、pgAdmin は PostgreSQL の管理ツールです。pgAdmin は SQL を定期的に実行するためのスケジューリングツールで、pgAdmin は PostgreSQL の管理ツールです。

```

##### SQL #####
# SQL # pgAgent ##### PostgreSQL ##### pgAgent #
##### pgAgent #####
database##### database#####
#####
##### database##### PostgreSQL #####
# SQL ##### PostgreSQL #####

```

```

##### pgAgent #####
## Windows ##### DOS ##### pgAgent #####
## Linux ##### shell #####

```

[illegible][illegible]

```

pgAgent pgAgent pgAgent
pgAgent

```

```

##### host
agent ##### pgAgent #####
#####

```

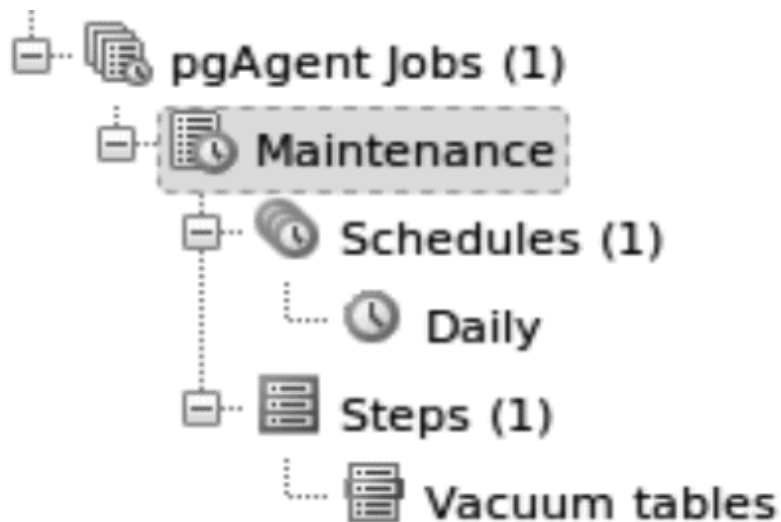


```

pgAgent
pgAgent schema
postgres pgAgent
PostgreSQL
pgAgent
PostgreSQL pgAgent

```

□□□□□□□□□□ 4-20 □□□



4-20 pgAdmin 查看 pgAgent 结构

4.5.3 查看 pgAgent 结构

通过 SQL 语句可以查看 pgAgent 结构，以下语句可以查看 pgAgent 结构，运行该语句后 postgres 显示结果 4-3 所示。

图 4-3 查看 pgAgent 结构

```

SELECT c.relname As table_name, d.description
FROM
    pg_class As c INNER JOIN
    pg_namespace n ON n.oid = c.relnamespace INNER JOIN
    pg_description As d ON d.objoid = c.oid AND d.objsubid = 0
WHERE n.nspname = 'pgagent'
ORDER BY c.relname;

```

| table_name | description |
|----------------|-------------------------|
| pga_job | Job main entry |
| pga_jobagent | Active job agents |
| pga_jobclass | Job classification |
| pga_joblog | Job run logs. |
| pga_jobstep | Job step to be executed |
| pga_jobsteplog | Job step run logs. |

pgAdmin pgAgent

4-4

```
SELECT j.jobname, s.jstname, l.jslstart,l.jslduration, l.jsloutput
FROM
    pgagent.pga_jobsteplog As l INNER JOIN
    pgagent.pga_jobstep As s ON s.jstid = l.jsljstid INNER JOIN
    pgagent.pga_job As j ON j.jobid = s.jstjobid
WHERE jslstart > CURRENT_DATE
ORDER BY j.jobname, s.jstname, l.jslstart DESC;
```


```

pgAgent shell
jsloutput shell

```



```

 Windows
pgAgent
shell
"bug"

```

5

PostgreSQL
PostgreSQL
JSON/XML
PostgreSQL
PostgreSQL PostgreSQL

PostgreSQL 数据库的序列



PostgreSQL 数据库的序列 $f(x)$ 函数“序列”数据库的序列
序列 + 序列 - 序列 * 序列 / 序列数据库的序列
序列数据库的序列数据库的序列数据库的序列数据库的序列
数据库的序列

5.1 序列

PostgreSQL 数据库的序列数据库的序列 serial 序列
数据库的序列

5.1.1 serial 序列

serial 序列 bigserial 序列数据库的序列
数据库的序列 serial 序列数据库的序列
数据库的序列 autonumber 序列数据库的序列
序列 serial 序列 PostgreSQL 数据库的序列
schema 序列 table_name_column_name_seq 序列数据库的序列
序列 serial 序列数据库的序列
序列


PostgreSQL 数据库的序列 pgAdmin 序列
ALTER SEQUENCE 序列数据库的序列
序列
increment 序列 CREATE SEQUENCE
序列数据库的序列
序列

integer 序列 bigint 序列
nextval(sequence_name) 序列 5-1 序列

5-1 序列


```
CREATE SEQUENCE s START 1;
CREATE TABLE stuff(id bigint DEFAULT nextval('s') PRIMARY KEY, name
text);
```



```
 serial serial
```

5.1.2 〇〇〇〇〇〇〇〇〇〇〇〇

PostgreSQL の generate_series 関数について
 generate_series 関数は、SQL 文 for 5-2 のように
 5-13 のように

□□ 5-2 □□□□□□□□□□□□□□□□

5-2 generate_series() 13

```
SELECT x FROM generate_series(1,51,13) As x;
```

| | |
|----|-------|
| x | ----- |
| 1 | |
| 14 | |
| 27 | |
| 40 | |

[illegible]

5.2 □□□□

PostgreSQL character char
character varying varchar text

char PostgreSQL
varchar text char
varchar text
varchar varchar text
text

varchar text PostgreSQL ODBC
varchar text varchar text
1GB TOAST

varchar text
“In Defense of Varchar(X)”

text varchar
“Using MS Access with PostgreSQL” varchar

5.2.1

lpad rpad rtrim ltrim trim
btrim substring | 5-3
5-4

5-3 lpad rpad

```
SELECT
  lpad('ab', 4, '0') As ab_lpad,
  rpad('ab', 4, '0') As ab_rpad,
  lpad('abcde', 4, '0') As ab_lpad_trunc; ❶
```

```
ab_lpad | ab_rpad | ab_lpad_trunc
-----+-----+-----
```

| | | |
|------|------|------|
| 00ab | ab00 | abcd |
|------|------|------|

❶ lpad 関数

trim 関数

5-4

```
SELECT
  a As a_before, trim(a) As a_trim, rtrim(a) As a_rt,
  i As i_before, ltrim(i, '0') As i_lt_0,
  rtrim(i, '0') As i_rt_0, trim(i, '0') As i_t_0
FROM (
  SELECT repeat(' ', 4) || i || repeat(' ', 4) As a, '0' || i
  As i
  FROM generate_series(0, 200, 50) As i
) As x;
```

| a_before | a_trim | a_rt | i_before | i_lt_0 | i_rt_0 | i_t_0 |
|----------|--------|------|----------|--------|--------|-------|
| 0 | 0 | 0 | 00 | | | |
| 50 | 50 | 50 | 050 | 50 | 05 | 5 |
| 100 | 100 | 100 | 0100 | 100 | 01 | 1 |
| 150 | 150 | 150 | 0150 | 150 | 015 | 15 |
| 200 | 200 | 200 | 0200 | 200 | 02 | 2 |

string_agg 関数 3-14 5-26

5.2.2

PostgreSQL

split_part 関数 5-5

5-5

x

123

5-6

```
x
---
abc
123
z45
```

PostgreSQL 数据库的 back reference 在 PostgreSQL 中，使用“”和“”来引用。

5-7

X

(619) 730-6254

\\1 \2 正则表达式 \ 正则表达式
正则表达式E' PostgreSQL 正则表达式 \ 正则表达式

正则表达式 5-8 正则表达式 SQL 正则表达式

5-8 正则表达式

```
SELECT unnest(regexp_matches(
'Cell (619) 852-5083. Work (619)123-4567 , Casa 619-730-6254.
Bésame mucho.',
E'[(]{0,1}[0-9]{3}[)]-.[{0,1}[\s]{0,1}[0-9]{3}[-.]{0,1}[0-9]{4}',
'g')
) As x;


x
-----
(619) 852-5083
(619)123-4567
619-730-6254
(3 rows)
```

5-8 正则表达式

- [(]{0,1} 匹配 0 或 1 个 (
- [0-9]{3} 匹配 3 个数字
- [)]-.[{0,1} 匹配 0 或 1 个) - .
- [0-9]{4} 匹配 4 个数字
- `regexp_matches` 正则表达式函数，返回匹配结果。正则表达式 `flags` 参数，`g` 表示 `global`，即全局匹配。正则表达式 `flags` 参数，`gi` 表示 `global` 且 `ignore case`，即全局匹配且不区分大小写。正则表达式 `flags` 参数，`POSIX` 表示 `POSIX` 正则表达式。

- unnest □□□□□□□□□□□□□□



 00000000000000000000\\d 00 [0-9] 000000000000
000000000000000000000000000000000000

XXXXXXXXXXXXXXXX substring XXXXXX 5-9 XXX

□□ 5-9 □□□□□□□□□□□□□□

```
SELECT substring(
'Cell (619) 852-5083. Work (619)123-4567 , Casa 619-730-6254.
Besame mucho.'
from E'[(]{0,1}[0-9]{3}[)]-.]{0,1}[\s]{0,1}[0-9]{3}[-.]{0,1}[0-9]{4}')
As x;

      x
-----
(619) 852-5083
(1 row)
```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX SIMILAR TO ~XXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX description XX

```
SELECT description
FROM mytable
WHERE description ~
E'([(){0,1}[0-9]{3}[])-.]{0,1}[\\s]{0,1}[0-9]{3}[-.]{0,1}[0-9]{4}';
```

5.3 □□□□

PostgreSQL PostgreSQL
interval PostgreSQL
5.5

PostgreSQL 9.1.2 支持的范围与 ANSI SQL 兼容，但 PostgreSQL 9.1.2 支持的范围与 Oracle 数据库 SQL Server 数据库 MySQL 数据库

PostgreSQL 9.1.2 支持的范围与 ANSI SQL 兼容，但 PostgreSQL 9.1.2 支持的范围与 Oracle 数据库 SQL Server 数据库 MySQL 数据库

date

date

time [precision] time without time zone

time [precision]

timestamp [precision] timestamp without time zone

timestamp [precision] timestamp without time zone

timestamp [precision] timestamp with time zone

timestamp [precision] timestamp with time zone UTC 时间戳
timestamp [precision] timestamp with time zone / 时间 / 时间戳
timestamp [precision] timestamp with time zone timestampz 时间戳
PostgreSQL 9.1.2 支持的范围与 ANSI SQL 兼容，但 PostgreSQL 9.1.2 支持的范围与 Oracle 数据库 SQL Server 数据库 MySQL 数据库

timetz [precision] time with time zone

timestamp [precision] timestamp with time zone
timestamp [precision] timestamp with time zone / 时间 / 时间戳
timestamp [precision] timestamp with time zone timestampz 时间戳
1970 年 1 月 1 日 00:00:00 时间戳

interval

interval 666 interval

tsrange

timestamp with no timezone '[2012-01-01 14:00 2012-01-01 15:00)':::tsrange 14:00 15:00 PostgreSQL “”

tstzrange

timestamp with timezone

daterange

5.3.1

PostgreSQL PostgreSQL 2012-2-14 18:08:00-8 -8 UTC 8 PostgreSQL

(1) 2012-02-14 18:08:00-8 UTC 2012-02-15 04:08:00-0

(2) UTC

PostgreSQL

(1)

(2) UTC America/New_York UTC -5

(3) UTC 2012-02-15 16:08:00
-5 2012-02-15 21:08:00

(4) 2012-02-15 21:08:00

```

##### PostgreSQL ##### UTC #####
##### PostgreSQL #####
##### UTC #####
#####
#####
#####
#####
#####

```

```

0000000000000000000000000000000000000000000000000000000
0000000000 timetz 0000000000000000000000000000000000000000
00000000000000000000000000000000000007 0000000000000000
00000000000000000000000007 00000000000000000000000004 00000000
00000000000000000000000004 00000000000000000000000000000000
000000000000000000000000000000000000000000000000000000

```

[illegible]

2012 年 3 月 11 日 1 时 50 分
 2012 年 3 月 11 日 3 时 10 分
 20 分

```
SELECT '2012-03-11 3:10 AM America/Los_Angeles'::timestampz
- '2012-03-11 1:50 AM America/Los Angeles'::timestampz;
```

20 1 20

```
SELECT '2012-03-11 3:10 AM'::timestamp- '2012-03-11 1:50 AM'::timestamp;
```

5-10 UTC

5-10

```
SELECT '2012-02-28 10:00 PM America/Los_Angeles'::timestampz;  
2012-02-29 01:00:00-05
```

5-11

5-11

```
SELECT '2012-02-28 10:00 PM America/Los_Angeles'::timestampz  
AT TIME ZONE 'Europe/Paris';  
2012-02-29 07:00:00
```

2012-02-28 10:00 p.m. UTC

http://en.wikipedia.org/wiki/Tz_database

5.3.2

[illegible]

interval

□ □ □ □ □ - □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
-- 5-12 Overlaps
-- true ANSI SQL overlaps
OVERLAPS
OVERLAPS
BETWEEN BETWEEN
0 OVERLAPS
```

5-12 OVERLAPS

```
SELECT
    ('2012-10-25 10:00 AM'::timestamp, '2012-10-25 2:00
PM'::timestamp)
```

```

OVERLAPS
('2012-10-25 11:00 AM'::timestamp, '2012-10-26 2:00
PM'::timestamp) AS x,
('2012-10-25'::date, '2012-10-26'::date)
OVERLAPS
('2012-10-26'::date, '2012-10-27'::date) As y;

x |y
---+---
t |f

```

PostgreSQL 2012-10-25 11:00 AM 2:00 PM
 PostgreSQL 2012-10-25 11:00 AM 2:00 PM
 PostgreSQL 2012-10-25 11:00 AM 2:00 PM

generate_series 2012-10-25 11:00 AM 2:00 PM
 interval 2012-10-25 11:00 AM 2:00 PM

5-13 PostgreSQL ISO
 "yyyy-mm-dd" PostgreSQL
 ISO PostgreSQL
 PostgreSQL

5-13 generate_series() PostgreSQL

```

SELECT (dt - interval '1 day')::date As eom
FROM generate_series('2/1/2012', '6/30/2012', interval '1 month')
As dt;

```

```

eom
-----
2012-01-31
2012-02-29
2012-03-31
2012-04-30
2012-05-31

```

PostgreSQL date_part to_char 5-14 PostgreSQL

US/East 2

5-14

```
SELECT dt, date_part('hour',dt) As hr, to_char(dt,'HH12:MI AM') As
mn
FROM
generate_series(
    '2012-03-11 12:30 AM',
    '2012-03-11 3:00 AM',
    interval '15 minutes'
) As dt;
```

| dt | hr | mn |
|------------------------|----|----------|
| 2012-03-11 00:30:00-05 | 0 | 12:30 AM |
| 2012-03-11 00:45:00-05 | 0 | 12:45 AM |
| 2012-03-11 01:00:00-05 | 1 | 01:00 AM |
| 2012-03-11 01:15:00-05 | 1 | 01:15 AM |
| 2012-03-11 01:30:00-05 | 1 | 01:30 AM |
| 2012-03-11 01:45:00-05 | 1 | 01:45 AM |
| 2012-03-11 03:00:00-04 | 3 | 03:00 AM |

generate_series timesatamp tz timestamp

5.4

PostgreSQL IN ANY PostgreSQL PostgreSQL PostgreSQL integer integer[] character character[] PostgreSQL

5.4.1

ARRAY[2001, 2002, 2003] As yrs;

```
SELECT ARRAY[2001, 2002, 2003] As yrs;
```

array() returns an empty array

```
SELECT array(
SELECT DISTINCT date_part('year', log_ts)
FROM logs
ORDER BY date_part('year', log_ts)
);
```

array() returns an empty array

array() returns an empty array

```
SELECT '{Alex,Sonia}'::text[] As name, '{46,43}'::smallint[] As
age;
```

| name | age |
|--------------|---------|
| {Alex,Sonia} | {46,43} |

string_to_array() returns an array

5-15 string_to_array()

```
SELECT string_to_array('ca.ma.tx', '.') As estados;
```

| estados |
|------------|
| {CA,MA,TX} |

(1 row)

array_agg 関数を使用して、ログのタイムスタンプを日付ごとに集約する 5-16

5-16 array_agg 関数

```
SELECT array_agg(log_ts ORDER BY log_ts) As x
FROM logs
WHERE log_ts BETWEEN '2011-01-01'::timestampz AND '2011-01-15'::timestampz;
x
-----
{'2011-01-01', '2011-01-13', '2011-01-14'}
```

PostgreSQL 9.5 の array_agg 関数は、配列の要素を指定した順序で集約する。array_agg 関数は、配列の要素を指定した順序で集約する。array_agg 関数は、配列の要素を指定した順序で集約する。5-17

5-17 array_agg 関数

```
SELECT array_agg(f.t)
FROM ( VALUES ('{Alex,Sonia}'::text[]),
      ('{46,43}'::text[] ) ) As f(t);

array_agg
-----
{{Alex,Sonia},{46,43}}
(1 row)
```

array_agg 関数は、配列の要素を指定した順序で集約する。array_agg 関数は、配列の要素を指定した順序で集約する。array_agg 関数は、配列の要素を指定した順序で集約する。5-17

5.4.2 unnest 関数

unnest 関数は、配列の要素を指定した順序で集約する。unnest 関数は、配列の要素を指定した順序で集約する。unnest 関数は、配列の要素を指定した順序で集約する。5-18

例 5-18 unnest の使用例

```
SELECT unnest('{X0X,0X0,X0X}'::char(3)[]) As tic_tac_toe;

tic_tac_toe
---
X0X
0X0
X0X
```

例 5-18 SELECT の使用例 unnest の使用例 unnest の使用例
例 5-18 “unnest” の使用例 unnest の使用例 unnest の使用例

例 5-19 unnest の使用例 unnest の使用例 unnest の使用例 3 例
unnest の使用例 3 例

例 5-19 unnest の使用例

```
SELECT
unnest('{three,blind,mice}'::text[]) As t,
unnest('{1,2,3}'::smallint[]) As i;

t      |i
-----+--
three  |1
blind  |2
mice   |3
```

例 5-19 unnest の使用例 unnest の使用例 unnest の使用例 5-20
unnest

例 5-20 unnest の使用例

```
SELECT
unnest( '{blind,mouse}'::varchar[]) As v,
unnest('{1,2,3}'::smallint[]) As i;

v      |i
-----+--
blind  |1
mouse  |2
```



```
blind |3
mouse |1
blind |2
mouse |3
```

PostgreSQL 9.4 のunnest 関数は、配列をテーブル形式に変換する。 ¹ 配列 null に対して unnest 関数は FROM 句で 5-21 のように指定する。 PostgreSQL 9.4 のunnest 関数は 5-20 のように指定する。

¹ 配列が null である場合は、unnest 関数は空の配列を返す。

5-21 unnest 関数の使用例

```
SELECT * FROM unnest('{blind,mouse}'::text[], '{1,2,3}'::int[]) AS
f(t,i);
```

| t | i |
|--------|---|
| blind | 1 |
| mouse | 2 |
| <NULL> | 3 |

5.4.3 配列の操作

PostgreSQL の start:end 記法は、配列の要素を指定する。 2 から 4 までの要素を指定する。

```
SELECT fact_subcats[2:4] FROM census.lu_fact_types;
```

配列の要素を指定する記法は、|| で結合する。

```
SELECT fact_subcats[1:2] || fact_subcats[3:4] FROM
census.lu_fact_types;
```

配列の要素を指定する記法は、|| で結合する。

```
SELECT '{1,2,3}'::integer[] || 4 || 5;
```

결과값은 {1,2,3,4,5} 이다

5.4.4 배열의 인덱싱

PostgreSQL 배열의 인덱싱은 1부터 시작하며, 배열의 길이를 초과하는 인덱스는 NULL을 반환한다.

```
SELECT
    fact_subcats[1] AS primero,
    fact_subcats[array_upper(fact_subcats, 1)] As segundo
FROM census.lu_fact_types;
```

array_upper 함수는 배열의 마지막 인덱스를 반환한다. PostgreSQL 배열의 인덱싱은 1부터 시작하며, 배열의 길이를 초과하는 인덱스는 NULL을 반환한다.

5.4.5 배열의 비교

PostgreSQL 배열의 비교 연산자는 ||, <>, <, >, @>, <@, &&, &를 사용한다. GiST 및 GIN 인덱싱도 지원된다.

&& 연산자는 true 또는 false를 반환한다. 5-22 배열의 비교 연산자 "fact_subcats"를 사용하여 OCCUPANCY STATUS 또는 For rent를 검색한다.

예제 5-22 배열의 비교

```
SELECT fact_subcats
FROM census.lu_fact_types
WHERE fact_subcats && '{OCCUPANCY STATUS,For rent}'::varchar[];
```

fact_subcats

```
{S01,"OCCUPANCY STATUS","Total housing units"...}
{S02,"OCCUPANCY STATUS","Total housing units"...}
{S03,"OCCUPANCY STATUS","Total housing units"...}
{S10,"VACANCY STATUS","Vacant housing units","For rent"...}
(4 rows)
```

1. 在 SQL 中，使用 `IS` 关键字来检查一个值是否为 `NULL`。例如，`SELECT * FROM table WHERE column IS NULL;`。

2. 在 SQL 中，使用 `IN` 关键字来检查一个值是否在指定的集合中。例如，`SELECT * FROM table WHERE column IN (value1, value2, ...);`。

3. 在 SQL 中，使用 `LIKE` 关键字来检查一个字符串是否匹配指定的模式。例如，`SELECT * FROM table WHERE column LIKE 'pattern';`。

4. 在 SQL 中，使用 `REGEXP` 关键字来检查一个字符串是否匹配指定的正则表达式。例如，`SELECT * FROM table WHERE column REGEXP 'pattern';`。

5.23 正则表达式

```
SELECT '{1,2,3}'::int[] @> '{3,2}'::int[] AS contains;

contains
-----
t
(1 row)
SELECT '{1,2,3}'::int[] <@ '{3,2}'::int[] AS contained_by;

contained_by
-----
f
(1 row)
```

5.5 正则表达式

在 PostgreSQL 中，正则表达式用于匹配字符串。正则表达式由一系列字符组成，这些字符可以匹配字符串中的特定模式。正则表达式可以用于各种任务，例如验证输入、提取数据、替换文本等。

在 PostgreSQL 中，正则表达式使用 `~` 和 `~*` 运算符。例如，`SELECT * FROM table WHERE column ~ 'pattern';`。正则表达式可以用于匹配字符串中的特定模式，例如匹配数字、字母、特殊字符等。

- 区間 $(-2, 2]$ の要素は $-1, 0, 1, 2$
- 区間 $(-2, 2)$ の要素は $-1, 0, 1$
- 区間 $[-2, 2]$ の要素は $-2, -1, 0, 1, 2$

5.5.1 区間の表現

PostgreSQL では区間の型として `int4range`、`int8range`、`numrange`、`daterange`、`tsrange`、`tstzrange` が用意されている。

区間 $(-2, 2)$ の要素は `int4range`、`int8range`、`numrange`、`daterange`、`tsrange`、`tstzrange` のいずれでも表現できる。ただし、`int4range`、`int8range`、`numrange` は整数範囲、`daterange` は日付範囲、`tsrange` はタイムスタンプ範囲、`tstzrange` はタイムゾーン付きタイムスタンプ範囲を表す。PostgreSQL では区間の型は `int4range`、`int8range`、`numrange`、`daterange`、`tsrange`、`tstzrange` のいずれかで表現される。例えば、`(2014-1-5, 2014-2-1]` は `daterange` の型で表現され、`[2014-01-06, 2014-02-02)` は `tsrange` の型で表現される。

5.5.2 区間の操作

PostgreSQL では区間の型に対して様々な操作が用意されている。

`int4range`、`int8range`

区間の要素は整数である。

`numrange`

区間の要素は数値である。

`daterange`

区間の要素は日付である。

`tsrange`、`tstzrange`

区間の要素はタイムスタンプである。ただし、`tsrange` はタイムゾーン付きタイムスタンプ範囲を表す。例えば、`(2014-1-5, 2014-2-1]` は `daterange` の型で表現され、`[2014-01-06, 2014-02-02)` は `tsrange` の型で表現される。

PostgreSQL null infinity int4range (-2147483648,2147483647)

-infinity infinity

5.5.3

[] () 5-24

5-24

```
SELECT '[2013-01-05,2013-08-13]':daterange; ❶
SELECT '(2013-01-05,2013-08-13)':daterange; ❷
SELECT '(0,)':int8range; ❸
SELECT '(2013-01-05 10:00,2013-08-13 14:00)':tsrange; ❹

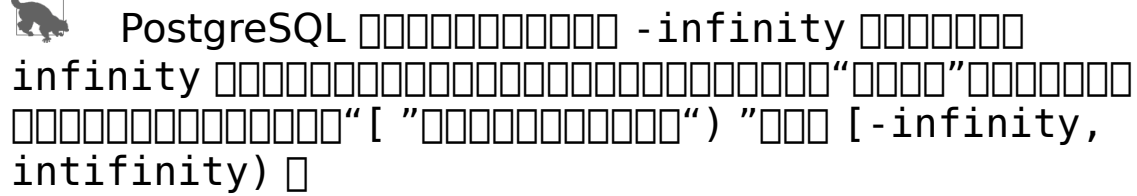
[2013-01-05,2013-08-14)
[2013-01-06,2013-08-14)
[1,)
("2013-01-05 10:00:00","2013-08-13 14:00:00"]
```

❶ 2013-01-05 2013-08-13

❷ 2013-01-05 2013-08-13

❸ 0

❹ 2013-01-05 10:00 AM 2013-08-13 2 PM



```
SELECT daterange('2013-01-05','infinity','[]');
```

5-25

```
CREATE TABLE employment (id serial PRIMARY KEY, employee
varchar(20),
period daterange);
CREATE INDEX ix_employment_period ON employment USING gist
(period);❶
INSERT INTO employment (employee,period)
VALUES
    ('Alex','[2012-04-24, infinity)'::daterange),
    ('Sonia','[2011-04-24, 2012-06-01)'::daterange),
    ('Leo','[2012-06-20, 2013-04-20)'::daterange),
    ('Regina','[2012-06-20, 2013-04-20)'::daterange);
```

5.5.5

이제 이 쿼리를 실행해보고, && 연산자 대신 @> 연산자를 사용하면
PostgreSQL은 “문자열”을

01. 문자열

이 쿼리를 실행해보고, && 연산자 대신 @> 연산자를 사용하면 true 또는
false 값을 5-26 쿼리에서 string_agg 함수
를 사용하여

5-26 쿼리

```
SELECT
    e1.employee,
    string_agg(DISTINCT e2.employee, ', ' ORDER BY
e2.employee) As colleagues
FROM employment As e1 INNER JOIN employment As e2
ON e1.period && e2.period
WHERE e1.employee <> e2.employee
GROUP BY e1.employee;
```

| employee | colleagues |
|----------|--------------------|
| Alex | Leo, Regina, Sonia |
| Leo | Alex, Regina |
| Regina | Alex, Leo |
| Sonia | Alex |

02. 날짜

이 쿼리를 실행해보고, @> 연산자를 사용하면 true 또는 false 값을 5-27 쿼리에서

5-27 쿼리

```
SELECT employee FROM employment WHERE period @> CURRENT_DATE
GROUP BY employee;
```

| employee |
|----------|
| Alex |


```

    },
    "phones":
    [
        {
            "type": "work",
            "number": "619-722-6719"
        },
        {
            "type": "cell",
            "number": "619-852-5083"
        }
    ]
},
"children":
[
    {
        "name": "Brandon",
        "gender": "M"
    },
    {
        "name": "Azaleah",
        "girl": true,
        "phones": []
    }
]
}'
);

```

5.6.2 JSON

JSON 5-29

5-29 JSON

```


SELECT person->'name' FROM persons;
SELECT person->'spouse' ->'parents' ->'father' FROM persons;

```

```

SELECT person#>array['spouse','parents','father'] FROM persons;

```

 JSON
 JSONB JSONB

PostgreSQL `json_extract_path` 関数は、JSON 文書から指定されたパスに沿って値を抽出します。この関数は、`key` として指定されたパスの各要素を、`json_extract_path_text` 関数で抽出した値として返します。

5.6.3 JSON

PostgreSQL は、JSON 文書をデータベースに格納するための `json` 型と、JSON 文書をテキスト形式に変換するための `json_text` 型を提供しています。

図 5-31 は、`row_to_json` 関数を使用して、図 5-28 の `persons` テーブルのデータを JSON 形式に変換するクエリを示しています。

図 5-31 JSON 形式に変換された PostgreSQL 9.3 の `persons` テーブルのデータ

```
SELECT row_to_json(f) As x
FROM (
    SELECT id, json_array_elements(person->'children')->>'name' As
    cname FROM persons
) As f;

           x
-----
{"id":1,"cname":"Brandon"}
{"id":1,"cname":"Azaleah"}
(2 rows)
```

`persons` テーブルのデータを JSON 形式に変換するクエリ

```
SELECT row_to_json(f) As jsoned_row FROM persons As f;
```

“JSON 形式に変換された PostgreSQL 9.3 の `persons` テーブルのデータ”という表は、PostgreSQL 9.3 の `array_agg` 関数と `array_to_json` 関数を使用して、JSON 形式に変換されたデータを生成しています。図 7-21 は、`json_agg` 関数の使用法を示しています。

5.6.4 JSON 形式に変換された `jsonb`

PostgreSQL 9.4 新增 jsonb 数据类型，jsonb 是 json 的超集，jsonb 数据类型是二进制存储的 json 数据，json 数据类型是文本存储的 json 数据。jsonb 数据类型是 json 数据类型的超集，jsonb 数据类型是 json 数据类型的超集，jsonb 数据类型是 json 数据类型的超集。

- json 数据类型是文本存储的 json 数据，jsonb 数据类型是二进制存储的 json 数据。jsonb 数据类型是 json 数据类型的超集，jsonb 数据类型是 json 数据类型的超集，jsonb 数据类型是 json 数据类型的超集。
- jsonb 数据类型是 json 数据类型的超集，jsonb 数据类型是 json 数据类型的超集。Michael Paquier 在“Manipulating jsonb data by abusing of key uniqueness”中介绍了 jsonb 数据类型的一些特性。
- jsonb 数据类型是 json 数据类型的超集，jsonb 数据类型是 json 数据类型的超集。GIN 索引在 PostgreSQL 6.3 版本中支持 json 数据类型，但 jsonb 数据类型在 PostgreSQL 9.4 版本中才被支持。

创建 persons 表，使用 jsonb 数据类型存储 person 数据。

```
CREATE TABLE persons_b (id serial PRIMARY KEY, person jsonb);
```

5-28 创建 persons_b 表

JSON 和 JSONB 数据类型在 PostgreSQL 9.4 版本中被引入。JSONB 数据类型是 JSON 数据类型的超集，JSONB 数据类型是 JSON 数据类型的超集。PostgreSQL 9.4 版本中，JSONB 数据类型是 JSON 数据类型的超集。

5-32 JSONB 和 JSON 数据类型

```
SELECT person As b FROM persons_b WHERE id = 1; ❶
SELECT person As j FROM persons WHERE id = 1; ❷
b
-----
-----
{"name": "Sonia",
"spouse": {"name": "Alex", "phones": [{"type": "work", "number":
"619-722-6719"},
{"type": "cell", "number": "619-852-5083"}],
```

```
"parents": {"father": "Rafael", "mother": "Ofelia"}},
"children": [{"name": "Brandon", "gender": "M"},
{"girl": true, "name": "Azaleah", "phones": []}]
(1 row)
```

j

```
-----
{
  "name": "Sonia",
  "spouse":
  {
    "name": "Alex",
    "parents":
    {
      "father": "Rafael",
      "mother": "Ofelia"
    },
    "phones":
    [
      {
        "type": "work",
        "number": "619-722-6719"+
      },
      {
        "type": "cell",
        "number": "619-852-5083"+
      }
    ]
  },
  "children":
  [
    {
      "name": "Brandon",
      "gender": "M"
    },
    {
      "name": "Azaleah",
      "girl": true,
      "phones": []
    }
  ]
}
(1 row)
```

❶ `jsonb` 데이터 형식은 JSON 데이터를 저장하는 데 사용됩니다. 이 형식은 데이터베이스에서 JSON 데이터를 저장하고 검색하는 데 유용합니다.

② json 数据类型

jsonb 与 json 数据类型的主要区别在于 jsonb 与 json 数据类型
存储方式不同 json 存储为文本 json_extract_path_text 与 json_each 函数
操作 jsonb 存储为二进制 jsonb_extract_path_text 与 jsonb_each 函数
操作二进制数据类型 5.6.2 数据类型 jsonb 数据类型
与 json_array_elements 函数 jsonb_array_elements 函数

jsonb 与 json 数据类型的主要区别在于 jsonb 与 json 数据类型
存储方式不同 json 存储为文本 json_extract_path_text 与 json_each 函数
操作 jsonb 存储为二进制 jsonb_extract_path_text 与 jsonb_each 函数
操作二进制数据类型 5.6.2 数据类型 jsonb 数据类型
与 json_array_elements 函数 jsonb_array_elements 函数

数据类型 jsonb 与 json 数据类型的主要区别在于 jsonb 与 json 数据类型
存储方式不同 json 存储为文本 json_extract_path_text 与 json_each 函数
操作 jsonb 存储为二进制 jsonb_extract_path_text 与 jsonb_each 函数
操作二进制数据类型 5.6.2 数据类型 jsonb 数据类型
与 json_array_elements 函数 jsonb_array_elements 函数

5-33 JSONB 数据类型

```
SELECT person->>'name' As name
FROM persons_b
WHERE person @> '{"children":[{"name":"Brandon"}]}';

name
-----
Sonia
```

数据类型 jsonb 与 json 数据类型的主要区别在于 jsonb 与 json 数据类型
存储方式不同 json 存储为文本 json_extract_path_text 与 json_each 函数
操作 jsonb 存储为二进制 jsonb_extract_path_text 与 jsonb_each 函数
操作二进制数据类型 5.6.2 数据类型 jsonb 数据类型
与 json_array_elements 函数 jsonb_array_elements 函数

```
CREATE INDEX ix_persons_jb_person_gin ON persons_b USING gin
(person);
```

数据类型 jsonb 与 json 数据类型的主要区别在于 jsonb 与 json 数据类型
存储方式不同 json 存储为文本 json_extract_path_text 与 json_each 函数
操作 jsonb 存储为二进制 jsonb_extract_path_text 与 jsonb_each 函数
操作二进制数据类型 5.6.2 数据类型 jsonb 数据类型
与 json_array_elements 函数 jsonb_array_elements 函数

5.6.5 数据类型 JSONB

PostgreSQL 9.5 数据类型 JSONB 数据类型的主要区别在于 jsonb 与 json 数据类型
存储方式不同 json 存储为文本 json_extract_path_text 与 json_each 函数
操作 jsonb 存储为二进制 jsonb_extract_path_text 与 jsonb_each 函数
操作二进制数据类型 5.6.2 数据类型 jsonb 数据类型
与 json_array_elements 函数 jsonb_array_elements 函数

PostgreSQL 9.5 8.5 JavaScript

jsonb 5-34 Gomez jsonb RETURNING RETURNING 7.2.10

5-34 JSONB ||

```
UPDATE persons_b
SET person = person || '{"address": "Somewhere in San Diego, CA"}'::jsonb
WHERE person @> '{"name": "Sonia"}'
RETURNING person;
profile
-----
{"name": "Sonia", ... "address": "Somewhere in San Diego, CA",
"children": ...}
(1 row)
UPDATE 1
```

JSONB "Somewhere in San Diego, CA"

5-35

5-35 JSONB -

```
UPDATE persons_b
SET person = person - 'address'
WHERE person @> '{"name": "Sonia"}';
```

JSONB #- Azaleah girl

5-36 JSONB #-

5.7.1 XML

xml PostgreSQL XML text XML text xml XML DTD XSD PostgreSQL XML XML 5-38 xml XML

5-38 XML

```
CREATE TABLE families (id serial PRIMARYKEY, profile xml);
INSERT INTO families(profile)
VALUES(
    '<family name="Gomez">
        <member><relation>padre</relation><name>Alex</name>
    </member>
        <member><relation>madre</relation><name>Sonia</name>
    </member>
        <member><relation>hijo</relation><name>Brandon</name>
    </member>
        <member><relation>hija</relation><name>Azaleah</name>
    </member>
    </family>');

```

XML XML check XML check 6.2.3 5-39 XML family relation XPath XPath XML

5-39 XML member relation

```
ALTER TABLE families ADD CONSTRAINT chk_has_relation
CHECK (xpath_exists('/family/member/relation', profile));

```

```
INSERT INTO families (profile) VALUES ('<family name="Hsu0be">
</family>');
```

ERROR: new row for relation "families" violates check constraint "chk_has_relation"
"families" violates constraint "chk_has_relation"

DTD 与 XSD 是 XML 文档的元数据，用于描述文档的结构和类型。PostgreSQL 支持 DTD 和 XSD 的验证。

5.7.2 XML

XML 文档的 XPath 表达式用于查询文档中的特定部分。XPath 表达式可以返回 XML 文档中的单个元素或元素列表。5-40 展示了如何使用 XPath 表达式来查询 XML 文档中的元素。

图 5-40 XML 查询

```
SELECT ordinality AS id, family,
       (xpath('/member/relation/text()', f))[1]::text As relation,
       (xpath('/member/name/text()', f))[1]::text As mem_name ❶
FROM (
    SELECT
        (xpath('/family/@name', profile))[1]::text As family, ❷
        f.ordinality, f.f
    FROM families, unnest(xpath('/family/member', profile))
WITH ORDINALITY AS f
) x; ❸
```

| id | family | relation | mem_name |
|----|--------|----------|----------|
| 1 | Gomez | padre | Alex |
| 2 | Gomez | madre | Sonia |
| 3 | Gomez | hijo | Brandon |
| 4 | Gomez | hija | Azaleah |

(4 rows)

❶ ① member ② relation ③ name ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱ ⑲ ⑳ ㉑ ㉒ ㉓ ㉔ ㉕ ㉖ ㉗ ㉘ ㉙ ㉚ ㉛ ㉜ ㉝ ㉞ ㉟ ㊱ ㊲ ㊳ ㊴ ㊵ ㊶ ㊷ ㊸ ㊹ ㊺ ㊻ ㊼ ㊽ ㊾ ㊿ ㏀ ㏁ ㏂ ㏃ ㏄ ㏅ ㏆ ㏇ ㏈ ㏉ ㏊ ㏋ ㏌ ㏍ ㏎ ㏏ ㏐ ㏑ ㏒ ㏓ ㏔ ㏕ ㏖ ㏗ ㏘ ㏙ ㏚ ㏛ ㏜ ㏝ ㏞ ㏟ ㏠ ㏡ ㏢ ㏣ ㏤ ㏥ ㏦ ㏧ ㏨ ㏩ ㏪ ㏫ ㏬ ㏭ ㏮ ㏯ ㏰ ㏱ ㏲ ㏳ ㏴ ㏵ ㏶ ㏷ ㏸ ㏹ ㏺ ㏻ ㏼ ㏽ ㏾ ㏿ 㐀 㐁 㐂 㐃 㐄 㐅 㐆 㐇 㐈 㐉 㐊 㐋 㐌 㐍 㐎 㐏 㐐 㐑 㐒 㐓 㐔 㐕 㐖 㐗 㐘 㐙 㐚 㐛 㐜 㐝 㐞 㐟 㐠 㐡 㐢 㐣 㐤 㐥 㐦 㐧 㐨 㐩 㐪 㐫 㐬 㐭 㐮 㐯 㐰 㐱 㐲 㐳 㐴 㐵 㐶 㐷 㐸 㐹 㐺 㐻 㐼 㐽 㐾 㐿 㑀 㑁 㑂 㑃 㑄 㑅 㑆 㑇 㑈 㑉 㑊 㑋 㑌 㑍 㑎 㑏 㑐 㑑 㑒 㑓 㑔 㑕 㑖 㑗 㑘 㑙 㑚 㑛 㑜 㑝 㑞 㑟 㑠 㑡 㑢 㑣 㑤 㑥 㑦 㑧 㑨 㑩 㑪 㑫 㑬 㑭 㑮 㑯 㑰 㑱 㑲 㑳 㑴 㑵 㑶 㑷 㑸 㑹 㑺 㑻 㑼 㑽 㑾 㑿 㒀 㒁 㒂 㒃 㒄 㒅 㒆 㒇 㒈 㒉 㒊 㒋 㒌 㒍 㒎 㒏 㒐 㒑 㒒 㒓 㒔 㒕 㒖 㒗 㒘 㒙 㒚 㒛 㒜 㒝 㒞 㒟 㒠 㒡 㒢 㒣 㒤 㒥 㒦 㒧 㒨 㒩 㒪 㒫 㒬 㒭 㒮 㒯 㒰 㒱 㒲 㒳 㒴 㒵 㒶 㒷 㒸 㒹 㒺 㒻 㒼 㒽 㒾 㒿 㓀 㓁 㓂 㓃 㓄 㓅 㓆 㓇 㓈 㓉 㓊 㓋 㓌 㓍 㓎 㓏 㓐 㓑 㓒 㓓 㓔 㓕 㓖 㓗 㓘 㓙 㓚 㓛 㓜 㓝 㓞 㓟 㓠 㓡 㓢 㓣 㓤 㓥 㓦 㓧 㓨 㓩 㓪 㓫 㓬 㓭 㓮 㓯 㓰 㓱 㓲 㓳 㓴 㓵 㓶 㓷 㓸 㓹 㓺 㓻 㓼 㓽 㓾 㓿 㔀 㔁 㔂 㔃 㔄 㔅 㔆 㔇 㔈 㔉 㔊 㔋 㔌 㔍 㔎 㔏 㔐 㔑 㔒 㔓 㔔 㔕 㔖 㔗 㔘 㔙 㔚 㔛 㔜 㔝 㔞 㔟 㔠 㔡 㔢 㔣 㔤 㔥 㔦 㔧 㔨 㔩 㔪 㔫 㔬 㔭 㔮 㔯 㔰 㔱 㔲 㔳 㔴 㔵 㔶 㔷 㔸 㔹 㔺 㔻 㔼 㔽 㔾 㔿 㕀 㕁 㕂 㕃 㕄 㕅 㕆 㕇 㕈 㕉 㕊 㕋 㕌 㕍 㕎 㕏 㕐 㕑 㕒 㕓 㕔 㕕 㕖 㕗 㕘 㕙 㕚 㕛 㕜 㕝 㕞 㕟 㕠 㕡 㕢 㕣 㕤 㕥 㕦 㕧 㕨 㕩 㕪 㕫 㕬 㕭 㕮 㕯 㕰 㕱 㕲 㕳 㕴 㕵 㕶 㕷 㕸 㕹 㕺 㕻 㕼 㕽 㕾 㕿 㖀 㖁 㖂 㖃 㖄 㖅 㖆 㖇 㖈 㖉 㖊 㖋 㖌 㖍 㖎 㖏 㖐 㖑 㖒 㖓 㖔 㖕 㖖 㖗 㖘 㖙 㖚 㖛 㖜 㖝 㖞 㖟 㖠 㖡 㖢 㖣 㖤 㖥 㖦 㖧 㖨 㖩 㖪 㖫 㖬 㖭 㖮 㖯 㖰 㖱 㖲 㖳 㖴 㖵 㖶 㖷 㖸 㖹 㖺 㖻 㖼 㖽 㖾 㖿 㗀 㗁 㗂 㗃 㗄 㗅 㗆 㗇 㗈 㗉 㗊 㗋 㗌 㗍 㗎 㗏 㗐 㗑 㗒 㗓 㗔 㗕 㗖 㗗 㗘 㗙 㗚 㗛 㗜 㗝 㗞 㗟 㗠 㗡 㗢 㗣 㗤 㗥 㗦 㗧 㗨 㗩 㗪 㗫 㗬 㗭 㗮 㗯 㗰 㗱 㗲 㗳 㗴 㗵 㗶 㗷 㗸 㗹 㗺 㗻 㗼 㗽 㗾 㗿 㘀 㘁 㘂 㘃 㘄 㘅 㘆 㘇 㘈 㘉 㘊 㘋 㘌 㘍 㘎 㘏 㘐 㘑 㘒 㘓 㘔 㘕 㘖 㘗 㘘 㘙 㘚 㘛 㘜 㘝 㘞 㘟 㘠 㘡 㘢 㘣 㘤 㘥 㘦 㘧 㘨 㘩 㘪 㘫 㘬 㘭 㘮 㘯 㘰 㘱 㘲 㘳 㘴 㘵 㘶 㘷 㘸 㘹 㘺 㘻 㘼 㘽 㘾 㘿 㙀 㙁 㙂 㙃 㙄 㙅 㙆 㙇 㙈 㙉 㙊 㙋 㙌 㙍 㙎 㙏 㙐 㙑 㙒 㙓 㙔 㙕 㙖 㙗 㙘 㙙 㙚 㙛 㙜 㙝 㙞 㙟 㙠 㙡 㙢 㙣 㙤 㙥 㙦 㙧 㙨 㙩 㙪 㙫 㙬 㙭 㙮 㙯 㙰 㙱 㙲 㙳 㙴 㙵 㙶 㙷 㙸 㙹 㙺 㙻 㙼 㙽 㙾 㙿 㚀 㚁 㚂 㚃 㚄 㚅 㚆 㚇 㚈 㚉 㚊 㚋 㚌 㚍 㚎 㚏 㚐 㚑 㚒 㚓 㚔 㚕 㚖 㚗 㚘 㚙 㚚 㚛 㚜 㚝 㚞 㚟 㚠 㚡 㚢 㚣 㚤 㚥 㚦 㚧 㚨 㚩 㚪 㚫 㚬 㚭 㚮 㚯 㚰 㚱 㚲 㚳 㚴 㚵 㚶 㚷 㚸 㚹 㚺 㚻 㚼 㚽 㚾 㚿 㜀 㜁 㜂 㜃 㜄 㜅 㜆 㜇 㜈 㜉 㜊 㜋 㜌 㜍 㜎 㜏 㜐 㜑 㜒 㜓 㜔 㜕 㜖 㜗 㜘 㜙 㜚 㜛 㜜 㜝 㜞 㜟 㜠 㜡 㜢 㜣 㜤 㜥 㜦 㜧 㜨 㜩 㜪 㜫 㜬 㜭 㜮 㜯 㜰 㜱 㜲 㜳 㜴 㜵 㜶 㜷 㜸 㜹 㜺 㜻 㜼 㜽 㜾 㜿 㝀 㝁 㝂 㝃 㝄 㝅 㝆 㝇 㝈 㝉 㝊 㝋 㝌 㝍 㝎 㝏 㝐 㝑 㝒 㝓 㝔 㝕 㝖 㝗 㝘 㝙 㝚 㝛 㝜 㝝 㝞 㝟 㝠 㝡 㝢 㝣 㝤 㝥 㝦 㝧 㝨 㝩 㝪 㝫 㝬 㝭 㝮 㝯 㝰 㝱 㝲 㝳 㝴 㝵 㝶 㝷 㝸 㝹 㝺 㝻 㝼 㝽 㝾 㝿 㞀 㞁 㞂 㞃 㞄 㞅 㞆 㞇 㞈 㞉 㞊 㞋 㞌 㞍 㞎 㞏 㞐 㞑 㞒 㞓 㞔 㞕 㞖 㞗 㞘 㞙 㞚 㞛 㞜 㞝 㞞 㞟 㞠 㞡 㞢 㞣 㞤 㞥 㞦 㞧 㞨 㞩 㞪 㞫 㞬 㞭 㞮 㞯 㞰 㞱 㞲 㞳 㞴 㞵 㞶 㞷 㞸 㞹 㞺 㞻 㞼 㞽 㞾 㞿 㟀 㟁 㟂 㟃 㟄 㟅 㟆 㟇 㟈 㟉 㟊 㟋 㟌 㟍 㟎 㟏 㟐 㟑 㟒 㟓 㟔 㟕 㟖 㟗 㟘 㟙 㟚 㟛 㟜 㟝 㟞 㟟 㟠 㟡 㟢 㟣 㟤 㟥 㟦 㟧 㟨 㟩 㟪 㟫 㟬 㟭 㟮 㟯 㟰 㟱 㟲 㟳 㟴 㟵 㟶 㟷 㟸 㟹 㟺 㟻 㟼 㟽 㟾 㟿 㠀 㠁 㠂 㠃 㠄 㠅 㠆 㠇 㠈 㠉 㠊 㠋 㠌 㠍 㠎 㠏 㠐 㠑 㠒 㠓 㠔 㠕 㠖 㠗 㠘 㠙 㠚 㠛 㠜 㠝 㠞 㠟 㠠 㠡 㠢 㠣 㠤 㠥 㠦 㠧 㠨 㠩 㠪 㠫 㠬 㠭 㠮 㠯 㠰 㠱 㠲 㠳 㠴 㠵 㠶 㠷 㠸 㠹 㠺 㠻 㠼 㠽 㠾 㠿 㡀 㡁 㡂 㡃 㡄 㡅 㡆 㡇 㡈 㡉 㡊 㡋 㡌 㡍 㡎 㡏 㡐 㡑 㡒 㡓 㡔 㡕 㡖 㡗 㡘 㡙 㡚 㡛 㡜 㡝 㡞 㡟 㡠 㡡 㡢 㡣 㡤 㡥 㡦 㡧 㡨 㡩 㡪 㡫 㡬 㡭 㡮 㡯 㡰 㡱 㡲 㡳 㡴 㡵 㡶 㡷 㡸 㡹 㡺 㡻 㡼 㡽 㡾 㡿 㢀 㢁 㢂 㢃 㢄 㢅 㢆 㢇 㢈 㢉 㢊 㢋 㢌 㢍 㢎 㢏 㢐 㢑 㢒 㢓 㢔 㢕 㢖 㢗 㢘 㢙 㢚 㢛 㢜 㢝 㢞 㢟 㢠 㢡 㢢 㢣 㢤 㢥 㢦 㢧 㢨 㢩 㢪 㢫 㢬 㢭 㢮 㢯 㢰 㢱 㢲 㢳 㢴 㢵 㢶 㢷 㢸 㢹 㢺 㢻 㢼 㢽 㢾 㢿 㣀 㣁 㣂 㣃 㣄 㣅 㣆 㣇 㣈 㣉 㣊 㣋 㣌 㣍 㣎 㣏 㣐 㣑 㣒 㣓 㣔 㣕 㣖 㣗 㣘 㣙 㣚 㣛 㣜 㣝 㣞 㣟 㣠 㣡 㣢 㣣 㣤 㣥 㣦 㣧 㣨 㣩 㣪 㣫 㣬 㣭 㣮 㣯 㣰 㣱 㣲 㣳 㣴 㣵 㣶 㣷 㣸 㣹 㣺 㣻 㣼 㣽 㣾 㣿 㤀 㤁 㤂 㤃 㤄 㤅 㤆 㤇 㤈 㤉 㤊 㤋 㤌 㤍 㤎 㤏 㤐 㤑 㤒 㤓 㤔 㤕 㤖 㤗 㤘 㤙 㤚 㤛 㤜 㤝 㤞 㤟 㤠 㤡 㤢 㤣 㤤 㤥 㤦 㤧 㤨 㤩 㤪 㤫 㤬 㤭 㤮 㤯 㤰 㤱 㤲 㤳 㤴 㤵 㤶 㤷 㤸 㤹 㤺 㤻 㤼 㤽 㤾 㤿 㥀 㥁 㥂 㥃 㥄 㥅 㥆 㥇 㥈 㥉 㥊 㥋 㥌 㥍 㥎 㥏 㥐 㥑 㥒 㥓 㥔 㥕 㥖 㥗 㥘 㥙 㥚 㥛 㥜 㥝 㥞 㥟 㥠 㥡 㥢 㥣 㥤 㥥 㥦 㥧 㥨 㥩 㥪 㥫 㥬 㥭 㥮 㥯 㥰 㥱 㥲 㥳 㥴 㥵 㥶 㥷 㥸 㥹 㥺 㥻 㥼 㥽 㥾 㥿 㦀 㦁 㦂 㦃 㦄 㦅 㦆 㦇 㦈 㦉 㦊 㦋 㦌 㦍 㦎 㦏 㦐 㦑 㦒 㦓 㦔 㦕 㦖 㦗 㦘 㦙 㦚 㦛 㦜 㦝 㦞 㦟 㦠 㦡 㦢 㦣 㦤 㦥 㦦 㦧 㦨 㦩 㦪 㦫 㦬 㦭 㦮 㦯 㦰 㦱 㦲 㦳 㦴 㦵 㦶 㦷 㦸 㦹 㦺 㦻 㦼 㦽 㦾 㦿 㧀 㧁 㧂 㧃 㧄 㧅 㧆 㧇 㧈 㧉 㧊 㧋 㧌 㧍 㧎 㧏 㧐 㧑 㧒 㧓 㧔 㧕 㧖 㧗 㧘 㧙 㧚 㧛 㧜 㧝 㧞 㧟 㧠 㧡 㧢 㧣 㧤 㧥 㧦 㧧 㧨 㧩 㧪 㧫 㧬 㧭 㧮 㧯 㧰 㧱 㧲 㧳 㧴 㧵 㧶 㧷 㧸 㧹 㧺 㧻 㧼 㧽 㧾 㧿 㨀 㨁 㨂 㨃 㨄 㨅 㨆 㨇 㨈 㨉 㨊 㨋 㨌 㨍 㨎 㨏 㨐 㨑 㨒 㨓 㨔 㨕 㨖 㨗 㨘 㨙 㨚 㨛 㨜 㨝 㨞 㨟 㨠 㨡 㨢 㨣 㨤 㨥 㨦 㨧 㨨 㨩 㨪 㨫 㨬 㨭 㨮 㨯 㨰 㨱 㨲 㨳 㨴 㨵 㨶 㨷 㨸 㨹 㨺 㨻 㨼 㨽 㨾 㨿 㩀 㩁 㩂 㩃 㩄 㩅 㩆 㩇 㩈 㩉 㩊 㩋 㩌 㩍 㩎 㩏 㩐 㩑 㩒 㩓 㩔 㩕 㩖 㩗 㩘 㩙 㩚 㩛 㩜 㩝 㩞 㩟 㩠 㩡 㩢 㩣 㩤 㩥 㩦 㩧 㩨 㩩 㩪 㩫 㩬 㩭 㩮 㩯 㩰 㩱 㩲 㩳 㩴 㩵 㩶 㩷 㩸 㩹 㩺 㩻 㩼 㩽 㩾 㩿 㪀 㪁 㪂 㪃 㪄 㪅 㪆 㪇 㪈 㪉 㪊 㪋 㪌 㪍 㪎 㪏 㪐 㪑 㪒 㪓 㪔 㪕 㪖 㪗 㪘 㪙 㪚 㪛 㪜 㪝 㪞 㪟 㪠 㪡 㪢 㪣 㪤 㪥 㪦 㪧 㪨 㪩 㪪 㪫 㪬 㪭 㪮 㪯 㪰 㪱 㪲 㪳 㪴 㪵 㪶 㪷 㪸 㪹 㪺 㪻 㪼 㪽 㪾 㪿 㫀 㫁 㫂 㫃 㫄 㫅 㫆 㫇 㫈 㫉 㫊 㫋 㫌 㫍 㫎 㫏 㫐 㫑 㫒 㫓 㫔 㫕 㫖 㫗 㫘 㫙 㫚 㫛 㫜 㫝 㫞 㫟 㫠 㫡 㫢 㫣 㫤 㫥 㫦 㫧 㫨 㫩 㫪 㫫 㫬 㫭 㫮 㫯 㫰 㫱 㫲 㫳 㫴 㫵 㫶 㫷 㫸 㫹 㫺 㫻 㫼 㫽 㫾 㫿 㬀 㬁 㬂 㬃 㬄 㬅 㬆 㬇 㬈 㬉 㬊 㬋 㬌 㬍 㬎 㬏 㬐 㬑 㬒 㬓 㬔 㬕 㬖 㬗 㬘 㬙 㬚 㬛 㬜 㬝 㬞 㬟 㬠 㬡 㬢 㬣 㬤 㬥 㬦 㬧 㬨 㬩 㬪 㬫 㬬 㬭 㬮 㬯 㬰 㬱 㬲 㬳 㬴 㬵 㬶 㬷 㬸 㬹 㬺 㬻 㬼 㬽 㬾 㬿 㭀 㭁 㭂 㭃 㭄 㭅 㭆 㭇 㭈 㭉 㭊 㭋 㭌 㭍 㭎 㭏 㭐 㭑 㭒 㭓 㭔 㭕 㭖 㭗 㭘 㭙 㭚 㭛 㭜 㭝 㭞 㭟 㭠 㭡 㭢 㭣 㭤 㭥 㭦 㭧 㭨 㭩 㭪 㭫 㭬 㭭 㭮 㭯 㭰 㭱 㭲 㭳 㭴 㭵 㭶 㭷 㭸 㭹 㭺 㭻 㭼 㭽 㭾 㭿 㮀 㮁 㮂 㮃 㮄 㮅 㮆 㮇 㮈 㮉 㮊 㮋 㮌 㮍 㮎 㮏 㮐 㮑 㮒 㮓 㮔 㮕 㮖 㮗 㮘 㮙 㮚 㮛 㮜 㮝 㮞 㮟 㮠 㮡 㮢 㮣 㮤 㮥 㮦 㮧 㮨 㮩 㮪 㮫 㮬 㮭 㮮 㮯 㮰 㮱 㮲 㮳 㮴 㮵 㮶 㮷 㮸 㮹 㮺 㮻 㮼 㮽 㮾 㮿 㯀 㯁 㯂 㯃 㯄 㯅 㯆 㯇 㯈 㯉 㯊 㯋 㯌 㯍 㯎 㯏 㯐 㯑 㯒 㯓 㯔 㯕 㯖 㯗 㯘 㯙 㯚 㯛 㯜 㯝 㯞 㯟 㯠 㯡 㯢 㯣 㯤 㯥 㯦 㯧 㯨 㯩 㯪 㯫 㯬 㯭 㯮 㯯 㯰 㯱 㯲 㯳 㯴 㯵 㯶 㯷 㯸 㯹 㯺 㯻 㯼 㯽 㯾 㯿 㰀 㰁 㰂 㰃 㰄 㰅 㰆 㰇 㰈 㰉 㰊 㰋 㰌 㰍 㰎 㰏 㰐 㰑 㰒 㰓 㰔 㰕 㰖 㰗 㰘 㰙 㰚 㰛 㰜 㰝 㰞 㰟 㰠 㰡 㰢 㰣 㰤 㰥 㰦 㰧 㰨 㰩 㰪 㰫 㰬 㰭 㰮 㰯 㰰 㰱 㰲 㰳 㰴 㰵 㰶 㰷 㰸 㰹 㰺 㰻 㰼 㰽 㰾 㰿 㱀 㱁 㱂 㱃 㱄 㱅 㱆 㱇 㱈 㱉 㱊 㱋 㱌 㱍 㱎 㱏 㱐 㱑 㱒 㱓 㱔 㱕 㱖 㱗 㱘 㱙 㱚 㱛 㱜 㱝 㱞 㱟 㱠 㱡 㱢 㱣 㱤 㱥 㱦 㱧 㱨 㱩 㱪 㱫 㱬 㱭 㱮 㱯 㱰 㱱 㱲 㱳 㱴 㱵 㱶 㱷 㱸 㱹 㱺 㱻 㱼 㱽 㱾 㱿 㲀 㲁 㲂 㲃 㲄 㲅 㲆 㲇 㲈 㲉 㲊 㲋 㲌 㲍 㲎 㲏 㲐 㲑 㲒 㲓 㲔 㲕 㲖 㲗 㲘 㲙 㲚 㲛 㲜 㲝 㲞 㲟 㲠 㲡 㲢 㲣 㲤 㲥 㲦 㲧 㲨 㲩 㲪 㲫 㲬 㲭 㲮 㲯 㲰 㲱 㲲 㲳 㲴 㲵 㲶 㲷 㲸 㲹 㲺 㲻 㲼 㲽 㲾 㲿 㳀 㳁 㳂 㳃 㳄 㳅 㳆 㳇 㳈 㳉 㳊 㳋 㳌 㳍 㳎 㳏 㳐 㳑 㳒 㳓 㳔 㳕 㳖 㳗 㳘 㳙 㳚 㳛 㳜 㳝 㳞 㳟 㳠 㳡 㳢 㳣 㳤 㳥 㳦 㳧 㳨 㳩 㳪 㳫 㳬 㳭 㳮 㳯 㳰 㳱 㳲 㳳 㳴 㳵 㳶 㳷 㳸 㳹 㳺 㳻 㳼 㳽 㳾 㳿 㴀 㴁 㴂 㴃 㴄 㴅 㴆 㴇 㴈 㴉 㴊 㴋 㴌 㴍 㴎 㴏 㴐 㴑 㴒 㴓 㴔 㴕 㴖 㴗 㴘 㴙 㴚 㴛 㴜 㴝 㴞 㴟 㴠 㴡 㴢 㴣 㴤 㴥 㴦 㴧 㴨 㴩 㴪 㴫 㴬 㴭 㴮 㴯 㴰 㴱 㴲 㴳 㴴 㴵 㴶 㴷 㴸 㴹 㴺 㴻 㴼 㴽 㴾 㴿 㵀 㵁 㵂 㵃 㵄 㵅 㵆 㵇 㵈 㵉 㵊 㵋 㵌 㵍 㵎 㵏 㵐 㵑 㵒 㵓 㵔 㵕 㵖 㵗 㵘 㵙 㵚 㵛 㵜 㵝 㵞 㵟 㵠 㵡 㵢 㵣 㵤 㵥 㵦 㵧 㵨 㵩 㵪 㵫 㵬 㵭 㵮 㵯 㵰 㵱 㵲 㵳 㵴 㵵 㵶 㵷 㵸 㵹 㵺 㵻 㵼 㵽 㵾 㵿 㶀 㶁 㶂 㶃 㶄 㶅 㶆 㶇 㶈 㶉 㶊 㶋 㶌 㶍 㶎 㶏 㶐 㶑 㶒 㶓 㶔 㶕 㶖 㶗 㶘 㶙 㶚 㶛 㶜 㶝 㶞 㶟 㶠 㶡 㶢 㶣 㶤 㶥 㶦 㶧 㶨 㶩 㶪 㶫 㶬 㶭 㶮 㶯 㶰 㶱 㶲 㶳 㶴 㶵 㶶 㶷 㶸 㶹 㶺 㶻 㶼 㶽 㶾 㶿 㷀 㷁 㷂 㷃 㷄 㷅 㷆 㷇 㷈 㷉 㷊 㷋 㷌 㷍 㷎 㷏 㷐 㷑 㷒 㷓 㷔 㷕 㷖 㷗 㷘 㷙 㷚 㷛 㷜 㷝 㷞 㷟 㷠 㷡 㷢 㷣 㷤 㷥 㷦 㷧 㷨 㷩 㷪 㷫 㷬 㷭 㷮 㷯 㷰 㷱 㷲 㷳 㷴 㷵 㷶 㷷 㷸 㷹 㷺 㷻 㷼 㷽 㷾 㷿 㸀 㸁 㸂 㸃 㸄 㸅 㸆 㸇 㸈 㸉 㸊 㸋 㸌 㸍 㸎 㸏 㸐 㸑 㸒 㸓 㸔 㸕 㸖 㸗 㸘 㸙 㸚 㸛 㸜 㸝 㸞 㸟 㸠 㸡 㸢 㸣 㸤 㸥 㸦 㸧 㸨 㸩 㸪 㸫 㸬 㸭 㸮 㸯 㸰 㸱 㸲 㸳 㸴 㸵 㸶 㸷 㸸 㸹 㸺 㸻 㸼 㸽 㸾 㸿 㹀 㹁 㹂 㹃 㹄 㹅 㹆 㹇 㹈 㹉 㹊 㹋 㹌 㹍 㹎 㹏 㹐 㹑 㹒 㹓 㹔 㹕 㹖 㹗 㹘 㹙 㹚 㹛 㹜 㹝 㹞 㹟 㹠 㹡 㹢 㹣 㹤 㹥 㹦 㹧 㹨 㹩 㹪 㹫 㹬 㹭 㹮 㹯 㹰 㹱 㹲 㹳 㹴 㹵 㹶 㹷 㹸 㹹 㹺 㹻 㹼 㹽 㹿 㺀 㺁 㺂 㺃 㺄 㺅 㺆 㺇 㺈 㺉 㺊 㺋 㺌 㺍 㺎 㺏 㺐 㺑 㺒 㺓 㺔 㺕 㺖 㺗 㺘 㺙 㺚 㺛 㺜 㺝 㺞 㺟 㺠 㺡 㺢 㺣 㺤 㺥 㺦 㺧 㺨 㺩 㺪 㺫 㺬 㺭 㺮 㺯 㺰 㺱 㺲 㺳 㺴 㺵 㺶 㺷 㺸 㺹 㺺 㺻 㺼 㺽 㺾 㺿 㻀 㻁 㻂 㻃 㻄 㻅 㻆 㻇 㻈 㻉 㻊 㻋 㻌 㻍 㻎 㻏 㻐 㻑 㻒 㻓 㻔 㻕 㻖 㻗 㻘 㻙 㻚 㻛 㻜 㻝 㻞 㻟 㻠 㻡 㻢 㻣 㻤 㻥 㻦 㻧 㻨 㻩 㻪 㻫 㻬 㻭 㻮 㻯 㻰 㻱 㻲 㻳 㻴 㻵 㻶 㻷 㻸 㻹 㻺 㻻 㻼 㻽 㻾 㻿 㼀 㼁 㼂 㼃 㼄 㼅 㼆 㼇 㼈 㼉 㼊 㼋 㼌 㼍 㼎 㼏 㼐 㼑 㼒 㼓 㼔 㼕 㼖 㼗 㼘 㼙 㼚 㼛 㼜 㼝 㼞 㼟 㼠 㼡 㼢 㼣 㼤 㼥 㼦 㼧 㼨 㼩 㼪 㼫 㼬 㼭 㼮 㼯 㼰 㼱 㼲 㼳 㼴 㼵 㼶 㼷 㼸 㼹 㼺 㼻 㼼 㼽 㼾 㼿 㽀 㽁 㽂 㽃 㽄 㽅 㽆 㽇 㽈 㽉 㽊 㽋 㽌 㽍 㽎 㽏 㽐 㽑 㽒 㽓 㽔 㽕 㽖 㽗 㽘 㽙 㽚 㽛 㽜 㽝 㽞 㽟 㽠 㽡 㽢 㽣 㽤 㽥 㽦 㽧 㽨 㽩 㽪 㽫 㽬 㽭 㽮 㽯 㽰 㽱 㽲 㽳 㽴 㽵 㽶 㽷 㽸 㽹 㽺 㽻 㽼 㽽 㽾 㽿 㿀 㿁 㿂 㿃 㿄 㿅 㿆 㿇 㿈 㿉 㿊 㿋 㿌 㿍 㿎 㿏 㿐 㿑 㿒 㿓 㿔 㿕 㿖 㿗 㿘 㿙 㿚 㿛 㿜 㿝 㿞 㿟 㿠 㿡 㿢 㿣 㿤 㿥 㿦 㿧 㿨 㿩 㿪 㿫 㿬 㿭 㿮 㿯 㿰 㿱 㿲 㿳 㿴 㿵 㿶 㿷 㿸 㿹 㿺 㿻 㿼 㿽 㿾 㿿 㸀 㸁 㸂 㸃 㸄 㸅 㸆 㸇 㸈 㸉 㸊 㸋 㸌 㸍 㸎 㸏 㸐 㸑 㸒 㸓 㸔 㸕 㸖 㸗 㸘 㸙 㸚 㸛 㸜 㸝 㸞 㸟 㸠 㸡 㸢

families profile

② COLUMNS XML

③ WITH ORDINALITY FOR ORDINALITY

④ ../ @name family name family/member @ XML name='a value'

⑤ XML PATH XML /family/member/ralation relation relation PATH

5.8

like ilike like soundex LGBT² “LGBT” “lesbian gay bisexual transgender”

² “”——

FTS full text search “” FTS PostgreSQL

FTS “FTS” love romance infatuation lust

love loving loved eat eats ate eaten

a the on that

FTS romance campus romance campus
FTS romance romance romance
FTS romance romance romance

5.8.1 FTS

PostgreSQL 10 FTS pg_catalog schema

SELECT cfname FROM pg_ts_config; psql \dF

| |
|------------|
| cfname |
| ----- |
| simple |
| danish |
| dutch |
| english |
| finnish |
| french |
| german |
| hungarian |
| italian |
| norwegian |
| portuguese |
| romanian |
| russian |
| spanish |
| swedish |
| turkish |

| |
|-----------|
| (16 rows) |
|-----------|

PostgreSQL 数据库“数据库”“数据库”

[illegible]

hunsPELL

```

hunspell_dicts hunspell .....
hunspell en_us

```

(1) □□□□□□□□□□

(2) `en_us.affix` `en_us.dict` PostgreSQL `share/ tsearch_data`

(3) `hunspell_en_us--*.sql` `hunspell_en_us.control` `PostgreSQL` `share/extension`

[illegible]

```
CREATE EXTENSION hunspell_en_us SCHEMA pg_catalog;
```

psql 5-42 hunspell

5-42 FTS hunspell

```
\dF+ english_hunspell;
```

```
Text search configuration "pg_catalog.english_hunspell"
Parser: "pg_catalog.default"
```

| Token | Dictionaries |
|-----------------|-------------------------------|
| asciihword | english_hunspell,english_stem |
| asciiword | english_hunspell,english_stem |
| email | simple |
| file | simple |
| float | simple |
| host | simple |
| hword | english_hunspell,english_stem |
| hword_asciipart | english_hunspell,english_stem |
| hword_numpart | simple |
| hword_part | english_hunspell,english_stem |
| int | simple |
| numhword | simple |
| numword | simple |
| sfloat | simple |
| uint | simple |
| url | simple |
| url_path | simple |
| version | simple |
| word | english_hunspell,english_stem |



FTS

5-43 PostgreSQL English hunspell

5-43 FTS English

```
\dF+ english;

Text search configuration "pg_catalog.english"
Parser: "pg_catalog.default"
Token      | Dictionaries
-----+-----
asciihword  | english_stem
asciiword   | english_stem
email       | simple
file        | simple
float       | simple
host        | simple
hword       | english_stem
hword_asciipart | english_stem
hword_numpart | simple
```

| | | |
|------------|--|--------------|
| hword_part | | english_stem |
| int | | simple |
| numhword | | simple |
| numword | | simple |
| sfloat | | simple |
| uint | | simple |
| url | | simple |
| url_path | | simple |
| version | | simple |
| word | | english_stem |

hunspell
 english_hunspell

```
SHOW default_text_search_config;
```

```
ALTER DATABASE postgresql_book
SET default_text_search_config = 'pg_catalog.english';
```

database

5.8.2 TSVector

FTS
 tsvector
 tsvector
 FTS
 lexeme
 tsvector
 tsvector

to_tsvector 函数将文本解析成文本搜索 (FTS) 索引格式

图 5-44 展示了 FTS 索引格式

图 5-44 展示了 FTS 索引格式

```
SELECT
  c.name,
  CASE
    WHEN c.name = 'default' THEN to_tsvector(f.t)
    ELSE to_tsvector(c.name::regconfig,f.t)
  END As vect
FROM (
  SELECT 'Just dancing in the rain. I like to dance.'::text) As
f(t), (
  VALUES ('default'),('english'),('english_hunspell'),
  ('simple')
) As c(name);
name          | vect
-----+-----
default       | 'danc':2,9 'like':7 'rain':5
english       | 'danc':2,9 'like':7 'rain':5
english_hunspell | 'dance':2,9 'dancing':2 'like':7 'rain':5
simple         | 'dance':9 'dancing':2 'i':6 'in':3 'just':1
'like':7
'rain':5 'the':4 'to':8
(4 rows)
```

图 5-44 展示了 FTS 索引格式。English 和 Hunspell 索引格式都包含 just 和 to 等词。Simple 索引格式包含 dancing 和 danc 和 dance。Simple 索引格式包含 dancing 和 danc 和 dance。

to_vector 函数将文本解析成文本搜索 (FTS) 索引格式。图 5-44 展示了 FTS 索引格式。图 5-44 展示了 FTS 索引格式。

database 函数将 FTS 索引格式解析成 tsvector 格式。图 5-44 展示了 FTS 索引格式。

データベース

データベースをpsqlで管理する film.sql を実行する

```
\encoding utf8;  
\i film.sql
```

データベース film に tsvector を追加する 5-45

図 5-45 tsvector を追加する

```
ALTER TABLE film ADD COLUMN fts tsvector;  
UPDATE film  
SET fts =  
    setweight(to_tsvector(COALESCE(title, '')), 'A') ||  
    setweight(to_tsvector(COALESCE(description, '')), 'B');  
CREATE INDEX ix_film_fts_gin ON film USING gin (fts);
```

図 5-45 title と description を結合して tsvector を作成する。GIN インデックスは GiST インデックスよりも高速である。6.3 節で詳しく説明する。

データベース film に fts を追加する。setweight を使用する。

weight は A, B, C, D のいずれかである。A は title, B は description, C は title と description の両方、D は title と description の両方である。5-45 図は title と description の両方を A と B の両方に設定する。title と description の両方を A と B の両方に設定する。

tsvector を作成する。title と description の両方を tsvector に追加する。title と description の両方を tsvector に追加する。

to_tsvector 関数は、テキストを tsvector 型に変換します。図 5-46 を参照。

図 5-46 to_tsvector 関数の使用例

```
CREATE TRIGGER trig_tsv_film_iu
BEFORE INSERT OR UPDATE OF title, description ON film FOR EACH ROW
EXECUTE PROCEDURE tsvector_update_trigger(fts,'pg_catalog.english',
title,description);
```

この例では、title と description の列に対して insert または update の操作が行われるたびに、tsvector_update_trigger 関数が呼び出されます。

5.8.3 TSQueries

FTS 検索には、tsquery 型の変数が必要です。PostgreSQL 9.6 では、to_tsquery 関数を使用して tsquery 型の変数を生成できます。

to_tsquery 関数は、テキストを tsquery 型に変換します。図 5-47 を参照。to_tsquery 関数は、plainto_tsquery 関数と phraseto_tsquery 関数と異なり、PostgreSQL 9.6 では使用できません。

図 5-47 to_tsquery 関数の使用例

図 5-47 to_tsquery 関数の使用例

図 5-47 to_tsquery 関数の使用例

```
SELECT to_tsquery('business & analytics');

to_tsquery
-----
```

```
('business' | 'busy') & 'analyt'
```

```
5-49 phraseto_tsquery tsquery
```

```

SELECT phraseto_tsquery('business analytics');

phraseto_tsquery
-----
'busi' <-> 'analyt'

SELECT phraseto_tsquery('english_hunspell','business analytics');

phraseto_tsquery
-----
'business' <-> 'analyt' | 'busy' <-> 'analyt'

```

tsquery 'business & analytics'::tsquery

tsquery || tsquery1
 || tsquery2 tsquery1
 tsquery2 & tsquery2
 tsquery1 tsquery2

5-50

5-50 tsquery

```

SELECT plainto_tsquery('business analyst') ||
phraseto_tsquery('data scientist');

tsquery
-----
'busi' & 'analyst' | 'data' <-> 'scientist'

SELECT plainto_tsquery('business analyst') &&
phraseto_tsquery('data scientist');

tsquery
-----
'busi' & 'analyst' & ('data' <-> 'scientist')

```


00 5-52 00000000000000000000 FTS 00

```
SELECT left(title,50) As title, left(description,50) as description
FROM film
WHERE fts @@ to_tsquery('hunter <4> (scientist | chef)') AND title
> '';
```

| title | description |
|------------------|--|
| ALASKA PHANTOM | A Fanciful Saga of a Hunter And a Pastry Chef |
| DAUGHTER MADIGAN | A Beautiful Tale of a Hunter And a Mad Scientist |

(2 rows)

00 5-52 000 hunter 0 scientist 00 chef 00000000000000 4 000

5.8.5 〇〇〇〇〇〇〇〇〇〇

```
FTS ts_rank ts_rank_cd  
ts_rank coverage density  
tsvector ts_rank_cd  
0  
tsvector  
ts_rank ts_rank  
ts_rank_cd A B C D 1.0 0.4 0.2  
0.1 5-53
```

□□ 5-53 □□□□□□□□

```
SELECT title, left(description,50) As description,  
       ts_rank(fts,ts)::numeric(10,3) AS r  
FROM film, to_tsquery('english','love & (wait | indian | mad)') AS  
ts  
WHERE fts @@ ts AND title > ''  
ORDER BY r DESC;  
  
title          | description
```

```
| r
-----+-----
+-----
INDIAN LOVE | A Insightful Saga of a Mad Scientist And a Mad Sci
| 0.999
LAWRENCE LOVE | A Fanciful Yarn of a Database Administrator And a
| 0.252
(2 rows)
```

title title
 1 0 5-54 5-53

5-54

```
SELECT
  left(title,40) As title,
  ts_rank('{0,0,0,1}':::numeric[],fts,ts)::numeric(10,3) AS r,
  ts_rank_cd('{0,0,0,1}':::numeric[],fts,ts)::numeric(10,3) As rcd
FROM film, to_tsquery('english', 'love & (wait | indian | mad )')
AS ts
WHERE fts @@ ts AND title > ''
ORDER BY r DESC;

title | r | rcd
-----+-----+-----
INDIAN LOVE | 0.991 | 1.000
LAWRENCE LOVE | 0.000 | 0.000
(2 rows)
```

rank 0 title
 tsquery



FTS
 Oleg Bartunov "Some FTS Tricks"
 to_tsquery('english','social & (science |
 scientist)') to_tsquery('social &
 (science| scientist)')

5.8.6 全文检索

PostgreSQL 10 提供了全文检索功能，通过 `tsvector` 和 `tsquery` 类型实现。在 `film` 表中，`fts` 列存储了电影的全文索引。通过 `strip` 函数可以将 `fts` 列的值转换为 `tsvector` 类型。图 5-55 展示了查询结果。

图 5-55 全文检索结果

```
SELECT fts
FROM film
WHERE film_id = 1;

'academi':1A 'battl':15B 'canadian':20B 'dinosaur':2A 'drama':5B
'epic':4B
'feminist':8B 'mad':11B 'must':14B 'rocki':21B 'scientist':12B
'teacher':17B

SELECT strip(fts)
FROM film
WHERE film_id = 1;

'academi' 'battl' 'canadian' 'dinosaur' 'drama' 'epic' 'feminist'
'mad'
'must' 'rocki' 'scientist' 'teacher'
```

通过 `strip` 函数可以将 `fts` 列的值转换为 `tsvector` 类型。图 5-55 展示了查询结果。

5.8.7 JSON/JSONB 数据类型

PostgreSQL 10 提供了 `json` 和 `jsonb` 数据类型，用于存储 JSON 数据。通过 `ts_headline` 和 `to_tsvector` 函数可以将 `json` 或 `jsonb` 类型的值转换为 `tsvector` 类型。图 5-56 展示了查询结果。

例 5-56 json/jsonb から tsvector を生成

```
SELECT to_tsvector(person)
FROM persons WHERE id=1;

to_tsvector
-----
'-5083':19 '-6719':13 '-722':12 '-852':18 '619':11,17 'alex':3
'azaleah':25
'brandon':21 'cell':15 'm':23 'ofelia':7 'rafael':5 'sonia':1
'work':9
(1 row)
```

例 5-56 は、json/jsonb から tsvector を生成する。person という person_b という json/jsonb から ts_headline を to_tsvector に渡して、FTS テキストを ts_headline に to_tsvector に渡して、json/jsonb から tsvector を生成する。

例 5-56 は、ts_headline を json/jsonb から HTML を生成する。例 5-57 は JSON から Rafael を生成する。

例 5-57 HTML から JSON を生成

```
SELECT ts_headline(person->'spouse'->'parents', 'rafael'::tsquery)
FROM persons_b WHERE id=1;

{"father": "<b>Rafael</b>", "mother": "Ofelia"}
(1 row)
```

例 5-57 は、HTML から JSON を生成する。

5.9 テキスト検索

composite record row
composite record

5.9.1

PostgreSQL
“turducken”³ 5-58
“”

³ turducken turkey-duck-chicken
——

5-58 “”

```
CREATE TABLE chickens (id integer PRIMARY KEY);
CREATE TABLE ducks (id integer PRIMARY KEY, chickens chickens[]);
CREATE TABLE turkeys (id integer PRIMARY KEY, ducks ducks[]);

INSERT INTO ducks VALUES (1, ARRAY[ROW(1)::chickens,
ROW(1)::chickens]);
INSERT INTO turkeys VALUES (1, array(SELECT d FROM ducks d));
```

ducks chickens chickens
chickens
chickens ducks ducks turkeys
chicken duck duck turkey
“”

turkeys

```
SELECT * FROM turkeys;

output
-----
id | ducks
---+-----
1  | {"(1,\"{(1),(1)}\\")"}
```

turkey chicken

```
UPDATE turkeys SET ducks[1].chickens[2] = ROW(3)::chickens
WHERE id = 1 RETURNING *;
```

output

```
-----
id | ducks
---+-----
 1 | {"(1,\"{(1),(3)}\")"}
```

RETURNING 7.2.10
 RETURNING

json jsonb

```
SELECT id, to_jsonb(ducks) AS ducks_jsonb
FROM turkeys;
```

```
id | ducks_jsonb
---+-----
 1 | [{"id": 1, "chickens": [{"id": 1}, {"id": 3}]}]
(1 row)
```

PostgreSQL ducks chickens
 chickens turkeys ducks ducks
 chickens drop CASCADE ducks
 chickens ducks chickens
 turkeys ducks

5.9.2

```
CREATE TYPE complex_number AS (r double precision, i double
precision);
```

□□□□□□□□□□□□□□□□

```
CREATE TABLE circuits (circuit_id serial PRIMARY KEY, ac_volt
complex_number);
```

□□□□□□□□□□□□□□□□

```
SELECT circuit_id, (ac_volt).* FROM circuits;
```

□□□□□□□□

```
SELECT circuit_id, (ac_volt).r, (ac_volt).i FROM circuits;
```



□□□□□□□□□□ ac_volt □□□□□□□□□□□□□□□□
PostgreSQL □□□□ FROM □□□□□□ ac_volt □□□□□□□□□□□□□□
□□□□□□□□□□ PostgreSQL □□□□□□□□□□

5.9.3 □□□□□□□□□□

□ ANSI SQL □□□□□□ NULL □□□□□□□□“□□□”□□ NULL=NULL □□□“□
□□”□□ NULL != NULL □□□□□□□□□□□□□□□□□□□□□□□□ NULL □□□
□□□□ IS NULL□IS NOT NULL □□ NOT□somevalue IS NULL□□□□□
□□□□□□□□□□□□□□□□something IS NULL □□ something IS NOT
NULL □□□□□□□□□□□□□□□□□□□□□□□□□□□□

PostgreSQL □□□ NULL □□□□□□ ANSI SQL □□□□□□□□□□□□□□□□ IS
NULL □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ NULL□□□□□□□□□□□□
□“□□□□”□□□□□□□□□□□ IS NOT NULL □□□□□□□□□□□□□□□□□□□□□□□□
□□□ NULL□□□

5.9.4 □□□□□□□□□□□□□□□□

complex_number 数据类型 + 数据类型 8 数据类型
PostgreSQL 使用“”数据类型

数据类型“”数据类型

5-59

5-59 complex_number

```
CREATE OR REPLACE FUNCTION add(complex_number, complex_number)
RETURNS complex_number AS
$$
    SELECT
        ((COALESCE(($1).r,0) + COALESCE(($2).r,0)),
        (COALESCE(($1).i,0) + COALESCE(($2).i,0)))::complex_number;
$$
language sql;
```

5-60

5-60 complex_number +

```
CREATE OPERATOR + (
    PROCEDURE = add,
    LEFTARG = complex_number,
    RIGHTARG = complex_number,
    COMMUTATOR = +
);
```

+

```
SELECT (1,2)::complex_number + (3,-10)::complex_number;
```

(4, -8)

PostgreSQL 10.4.1 (Ubuntu 10.4.1-0ubuntu0.16.04.1) 2016-10-13 14:14:14 UTC
complex_number integer add complex_number + complex_number
--

PostgreSQL 10.4.1 (Ubuntu 10.4.1-0ubuntu0.16.04.1) 2016-10-13 14:14:14 UTC
PostgreSQL 10.4.1 (Ubuntu 10.4.1-0ubuntu0.16.04.1) 2016-10-13 14:14:14 UTC
--

6 PostgreSQL 10.4.1

PostgreSQL 10.4.1 (Ubuntu 10.4.1-0ubuntu0.16.04.1) 2016-10-13 14:14:14 UTC
PostgreSQL 10.4.1 (Ubuntu 10.4.1-0ubuntu0.16.04.1) 2016-10-13 14:14:14 UTC
PostgreSQL 10.4.1 (Ubuntu 10.4.1-0ubuntu0.16.04.1) 2016-10-13 14:14:14 UTC
--

PostgreSQL 10.4.1 (Ubuntu 10.4.1-0ubuntu0.16.04.1) 2016-10-13 14:14:14 UTC
PostgreSQL 10.4.1 (Ubuntu 10.4.1-0ubuntu0.16.04.1) 2016-10-13 14:14:14 UTC
PostgreSQL 10.4.1 (Ubuntu 10.4.1-0ubuntu0.16.04.1) 2016-10-13 14:14:14 UTC
--

6.1 PostgreSQL 10.4.1

PostgreSQL 10.4.1 (Ubuntu 10.4.1-0ubuntu0.16.04.1) 2016-10-13 14:14:14 UTC
PostgreSQL 10.4.1 (Ubuntu 10.4.1-0ubuntu0.16.04.1) 2016-10-13 14:14:14 UTC

6.1.1 PostgreSQL 10.4.1

6-1 PostgreSQL 10.4.1 SQL 10.4.1

6-1 PostgreSQL 10.4.1

```
CREATE TABLE logs (  
  log_id serial PRIMARY KEY, ❶  
  user_name varchar(50), ❷
```

```
description text, ❸
log_ts timestamp with time zone NOT NULL DEFAULT current_timestamp
); ❹
CREATE INDEX idx_logs_log_ts ON logs USING btree (log_ts);
```

❶ serial 16bit 有符号整数 serial 32bit 有符号整数
 128bit 有符号整数 schema 128bit 有符号整数 serial 128bit 有符号整数
 128bit 有符号整数 serial 128bit 有符号整数
 128bit 有符号整数 bigserial 128bit 有符号整数

❷ varchar 变长字符串 character varying 变长字符串
 varchar 变长字符串 text 变长字符串

❸ text 变长字符串

❹ timestamp with time zone 带时区的timestamp timestampz 带时区的timestamp
 timestamp 不带时区的timestamp UTC 协调世界时
 5.3.1 节

PostgreSQL 10 支持 IDENTITY 序列 IDENTITY 序列

log_id 序列 serial IDENTITY 序列 serial
 IDENTITY 序列

```
DROP SEQUENCE logs_log_id_seq CASCADE;
ALTER TABLE logs
  ALTER COLUMN log_id ADD GENERATED BY DEFAULT AS IDENTITY;
```

1 序列

```
ALTER TABLE logs
  ALTER COLUMN log_id RESTART WITH 2000;
```

6-2 节 IDENTITY 序列 serial 序列

6-2 IDENTITY 序列


```
CREATE TABLE logs (
log_id int GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
user_name varchar(50),
description text,
log_ts timestamp with time zone NOT NULL DEFAULT current_timestamp
);
```

図 6-2 図 6-1 のスキーマ

図 6-2 は、図 6-1 のスキーマの IDENTITY と serial の違いを示しています。IDENTITY は、serial と異なり、明示的に IDENTITY と指定する必要があります。serial は、明示的に serial と指定する必要があります。IDENTITY は、明示的に IDENTITY と指定する必要があります。

図 6-2 は、図 6-1 のスキーマの serial と IDENTITY の違いを示しています。serial は、明示的に serial と指定する必要があります。IDENTITY は、明示的に IDENTITY と指定する必要があります。

6.1.2 図 6-2

PostgreSQL のスキーマは、図 6-2 のように定義されています。PostgreSQL のスキーマは、図 6-2 のように定義されています。PostgreSQL のスキーマは、図 6-2 のように定義されています。check 制約は、図 6-3 に示されています。

図 6-3 スキーマ

```
CREATE TABLE logs_2011 (PRIMARY KEY (log_id)) INHERITS (logs);
CREATE INDEX idx_logs_2011_log_ts ON logs_2011 USING btree(log_ts);
ALTER TABLE logs_2011
    ADD CONSTRAINT chk_y2011
CHECK (
    log_ts >= '2011-1-1'::timestampz AND log_ts < '2012-1-1'::timestampz
);
```

```
); ❶
```

❶ ❶ check ❶ 2011 check ❶
❶

PostgreSQL 9.5 ❶
❶

6.1.3 ❶

PostgreSQL 10 ❶
❶
DDL ❶

❶
❶

- ❶ CREATE TABLE .. PARTITION BY RANGE ..
❶
- ❶
❶
- ❶
- ❶
❶
- ❶
❶
- ❶
❶

❶ 6-1 ❶ logs ❶ 6-3
❶

❶ logs ❶

```
DROP TABLE IF EXISTS logs CASCADE;
```

PARTITION BY 6-4
6-1
PARTITION BY RANGE (log_ts);

6-4

```
CREATE TABLE logs (  
  log_id int GENERATED BY DEFAULT AS IDENTITY,  
  user_name varchar(50),  
  description text,  
  log_ts timestamp with time zone NOT NULL DEFAULT current_timestamp  
) PARTITION BY RANGE (log_ts);
```

FOR VALUES
check 6-5 6-3
FOR VALUES FROM INHERITS

6-5

```
CREATE TABLE logs_2011 PARTITION OF logs ①  
FOR VALUES FROM ('2011-1-1') TO ('2012-1-1') ②;  
CREATE INDEX idx_logs_2011_log_ts ON logs_2011 USING btree(log_ts);  
③  
ALTER TABLE logs_2011 ADD CONSTRAINT pk_logs_2011 PRIMARY KEY  
(log_id) ④;
```

① logs

②
CREATE TABLE

③④

```
INSERT INTO logs(user_name, description ) VALUES ('regina',  
'Sleeping');
```

データベースのパーティション

```
ERROR: no partition of relation "logs" found for row
DETAIL: Partition key of the failing row contains
(log_ts) = (2017-05-25 02:58:28.057101-04).
```

データベースのパーティション

```
CREATE TABLE logs_gt_2011 PARTITION OF logs
FOR VALUES FROM ('2012-1-1') TO (unbounded);
```

図 6-5 unbounded パーティションの作成

```
SELECT * FROM logs_gt_2011;
```

データベースのパーティション

図 6-6 パーティションの作成

図 6-6 パーティションの作成

```
EXPLAIN ANALYZE SELECT * FROM logs WHERE log_ts > '2017-05-01';

Append (cost=0.00..15.25 rows=140 width=162)
(actual time=0.008..0.009 rows=1 loops=1)
  -> Seq Scan on logs_gt_2011 (cost=0.00..15.25 rows=140
width=162)
(actual time=0.008..0.008 rows=1 loops=1)
    Filter: (log_ts > '2017-05-01 00:00:00-04'::timestamp with
time zone)
Planning time: 0.152 ms
Execution time: 0.022 ms
```

PostgreSQL 10 の PSQL

```

\d+ logs

Table "public.logs"
:
Partition key: RANGE (log_ts)
Partitions: logs_2011
            FOR VALUES FROM ('2011-01-01 00:00:00-05') TO ('2012-01-01
            00:00:00-05'),
            logs_gt_2011
            FOR VALUES FROM ('2012-01-01 00:00:00-05') TO
            (UNBOUNDED)

```

6.1.4 备份

PostgreSQL 9.1 引入了 UNLOGGED 表，它不记录 WAL（write-ahead log）数据。这导致在灾难恢复时，UNLOGGED 表的数据可能丢失。因此，对于重要的 UNLOGGED 表，建议定期备份。备份频率通常为 10 到 15 分钟。

PostgreSQL 提供了 pg_dump 工具，用于备份数据库。pg_dump 可以导出数据库的 SQL 语句，也可以导出二进制格式的数据。pg_dump 支持多种备份策略，包括全量备份和增量备份。

图 6-7 备份策略

```

CREATE UNLOGGED TABLE web_sessions (
    session_id text PRIMARY KEY,
    add_ts timestampz,
    upd_ts timestampz,
    session_state xml);

```

PostgreSQL 9.3 引入了 GiST（Generalized Search Tree）索引。GiST 索引支持多种数据类型，包括 JSON、B-Tree 和 GIN（Generalized Inverted Index）索引。GiST 索引在全文检索和空间查询中表现出色。

PostgreSQL 9.5 引入了 BRIN（Block Range Index）索引。BRIN 索引是一种基于范围的索引，适用于时间序列数据。BRIN 索引可以显著减少索引的大小，并提高查询性能。

```
ALTER TABLE some_table SET LOGGED;
```

6.1.5 TYPE OF

```
CREATE TYPE basic_user AS (user_name varchar(50), pwd varchar(10));
```

6-8

6-8

```
CREATE TABLE super_users OF basic_user (CONSTRAINT pk_su PRIMARY
KEY (user_name));
```

PostgreSQL

0000000000 6-8 0000 super_users 00000000000000000000
0000

```
ALTER TYPE basic_user ADD ATTRIBUTE phone varchar(10) CASCADE;
```

CASCADE

6.2 □□□□

PostgreSQL 10.5.1
check



6.2.1

PostgreSQL 10.5.1
6-9

6-9

```
SET search_path=census, public;  
ALTER TABLE facts ADD CONSTRAINT fk_facts_1 FOREIGN KEY  
(fact_type_id)  
REFERENCES lu_fact_types (fact_type_id) ❶ ON UPDATE CASCADE ON  
DELETE RESTRICT;  
❷  
CREATE INDEX fki_facts_1 ON facts (fact_type_id); ❸
```

❶ facts lu_fact_types
lu_fact_types fact_type_id fact
fact_type_id

❷ (1) lu_fact_type
fact_type_id fact fact_type_id
(2) fact
fact_type_id lu_fact_type
fact_type_id ON DELETE RESTRICT

❸ PostgreSQL

PostgreSQL 9.6 でのユニーク制約の追加

6.2.2 ユニーク制約

ユニーク制約は、テーブルの列に重複した値を許可しない制約です。ユニーク制約は、テーブルの列にユニークな値を要求します。ユニーク制約は、テーブルの列にユニークな値を要求します。ユニーク制約は、テーブルの列にユニークな値を要求します。

```
ALTER TABLE logs_2011 ADD CONSTRAINT uq UNIQUE (user_name, log_ts);
```

PostgreSQL 9.6 でのユニーク制約の追加 6.3.4

6.2.3 check制約

check制約は、テーブルの列に特定の値を許可しない制約です。check制約は、テーブルの列に特定の値を許可しない制約です。check制約は、テーブルの列に特定の値を許可しない制約です。check制約は、テーブルの列に特定の値を許可しない制約です。check制約は、テーブルの列に特定の値を許可しない制約です。

```
ALTER TABLE logs ADD CONSTRAINT chk CHECK (user_name = lower(user_name));
```

check制約は、テーブルの列に特定の値を許可しない制約です。check制約は、テーブルの列に特定の値を許可しない制約です。check制約は、テーブルの列に特定の値を許可しない制約です。check制約は、テーブルの列に特定の値を許可しない制約です。check制約は、テーブルの列に特定の値を許可しない制約です。

6.2.4 外制約

外制約は、テーブルの列に特定の値を許可しない制約です。外制約は、テーブルの列に特定の値を許可しない制約です。外制約は、テーブルの列に特定の値を許可しない制約です。外制約は、テーブルの列に特定の値を許可しない制約です。外制約は、テーブルの列に特定の値を許可しない制約です。

PostgreSQL 9.2 範囲データタイプをインストールする depesz の
記事“Waiting for 9.2 Range Data Types”を参考にインストール
する

範囲データタイプ GiST をインストールする B-ツリー GiST をインストール
する btree_gist をインストールする

インストールする範囲データタイプをインストールする
6-10 範囲データタイプをインストールする && 範囲データタイプをインストールする = 範囲
データタイプをインストールする

図 6-10 インストールする

```
CREATE TABLE schedules(id serial primary key, room int, time_slot  
tstzrange);  
ALTER TABLE schedules ADD CONSTRAINT ex_schedules  
EXCLUDE USING gist (room WITH =, time_slot WITH &&);
```

PostgreSQL インストールする

範囲データタイプをインストールする
block インストールする

```
CREATE TABLE room_blocks(block_id integer primary key, rooms  
int[]);
```

block インストールする GiST インストールする
6-11 図

図 6-11 block インストールする

```
CREATE EXTENSION IF NOT EXISTS intarray;  
ALTER TABLE room_blocks  
ADD CONSTRAINT ex_room_blocks_rooms
```

```
EXCLUDE USING gist(rooms WITH &&);
```

intarray 一维数组索引 int4 与 int8 二维 GiST 索引
intarray 二维数组索引 GiST 索引

6.3 索引

PostgreSQL 索引是数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分。

PostgreSQL 索引是数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分。

索引是数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分。



索引 schema 索引是数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分。

6.3.1 PostgreSQL 索引

索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分。

B-索引

B-索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分，索引是 PostgreSQL 数据库的重要组成部分。

BRIN 索引

BRIN (block range index) 是 PostgreSQL 9.4 新增的索引类型。它基于 B-tree 索引，但存储的是数据块的范围而不是具体的值。BRIN 索引适用于数据分布均匀且范围较大的数据。BRIN 索引的维护成本较低，因为它只需要存储少量的元数据。B-tree 索引在点查询和范围查询中表现良好，但在处理大量数据时，BRIN 索引可能更具优势。

GiST

GiST (generalized search tree) 是一种通用的搜索树索引。它支持多种数据类型，包括点、矩形、文本等。GiST 索引在空间查询和全文搜索中表现良好。它允许用户自定义索引函数，以适应不同的数据类型和查询需求。

GiST 索引可以用于多种数据类型，包括点、矩形、文本等。它支持范围查询和点查询。GiST 索引的维护成本较高，因为它需要存储大量的元数据。在空间查询和全文搜索中，GiST 索引通常比 B-tree 索引更具优势。

GIN

GIN (generalized inverted index) 是一种通用的倒排索引。它支持多种数据类型，包括 json、hstore、pg_trgm 等。GIN 索引在全文搜索和模糊匹配中表现良好。它允许用户自定义索引函数，以适应不同的数据类型和查询需求。GIN 索引的维护成本较高，因为它需要存储大量的元数据。在全文搜索和模糊匹配中，GIN 索引通常比 B-tree 索引更具优势。GIN 索引可以用于 json、hstore、pg_trgm 等数据类型。GIN 索引在全文搜索和模糊匹配中表现良好。GIN 索引的维护成本较高，因为它需要存储大量的元数据。在全文搜索和模糊匹配中，GIN 索引通常比 B-tree 索引更具优势。GIN 索引可以用于 json、hstore、pg_trgm 等数据类型。GIN 索引在全文搜索和模糊匹配中表现良好。GIN 索引的维护成本较高，因为它需要存储大量的元数据。在全文搜索和模糊匹配中，GIN 索引通常比 B-tree 索引更具优势。

SP-GiST

SP-GiST (space-partitioning trees) 是一种空间分区树索引。它是 GiST 的一种变体，专门用于空间数据。SP-GiST 索引在空间查询中表现良好，因为它可以有效地分区空间数据。PostgreSQL 支持 point 和 box 类型的空间数据。SP-GiST 索引的维护成本较高，因为它需要存储大量的元数据。在空间查询中，SP-GiST 索引通常比 B-tree 索引更具优势。

□ □ □ □

GiST と GIN の違いを比較する。GiST と GIN の違いは、
 PostgreSQL 10 のリリースノートに記載されている。
 PostgreSQL のリリースノートは、PostgreSQL 10 のリリースノート
 の B-10000 以降に記載されている。

□□ B-□□□□ GiST □ GIN □□

PostgreSQL 数据库索引类型包括 B-tree、GiST 和 GIN。B-tree 索引是最常用的索引类型，适用于大多数场景。GiST 索引适用于几何数据类型和全文索引。GIN 索引适用于数组类型和全文索引。全文索引用于对文本数据进行全文检索，支持复杂的查询条件。全文索引的创建和使用需要配置相应的全文搜索引擎，如 PostgreSQL 自带的全文搜索引擎或外部的搜索引擎如 Elasticsearch。

1 PostgreSQL の full-text 検索機能と tsvector と tsquery の関係

PostgreSQL の RUM インデックス
 VODKA の RUM と GIN インデックスを PostgreSQL 9.6 に移植する
 RUM インデックスの full-text インデックス化
 RUM インデックスの性能比較

pgroonga PostgreSQL 9.5
PostgreSQL 9.6
PostgreSQL-PGRoonga pgroonga
PGRoonga
PostgreSQL PGRoonga
ILIKE LIKE '%something%' pg_trgm
JSONB Linux/Unix Windows

6.3.2 索引

索引是数据库中的一种特殊的数据结构，它用于快速查找数据库中的记录。索引可以看作是一个指向数据库记录的指针。索引的创建和维护是数据库管理中的重要任务。

索引的创建可以使用 `CREATE INDEX` 语句。例如，创建一个名为 `idx_range` 的索引，范围从 `A` 到 `Z`。

PostgreSQL 索引的创建和使用。索引的创建可以使用 `CREATE INDEX` 语句。索引的维护可以使用 `VACUUM` 语句。索引的删除可以使用 `DROP INDEX` 语句。

图 6-12 B-索引的创建和使用

```
SELECT am.amname AS index_method, opc.opcname AS opclass_name,
opc.opcintype::regtype AS indexed_type, opc.opcdefault AS
is_default
FROM pg_am am INNER JOIN pg_opclass opc ON opc.opcmethod = am.oid
WHERE am.amname = 'btree'
ORDER BY index_method, indexed_type, opclass_name;
```

| index_method | opclass_name | indexed_type | is_default |
|--------------|---------------------|--------------|------------|
| btree | bool_ops | boolean | t |
| ⋮ | | | |
| btree | text_ops | text | t |
| btree | text_pattern_ops | text | f |
| btree | varchar_ops | text | f |
| btree | varchar_pattern_ops | text | f |
| ⋮ | | | |

图 6-12 索引的创建和使用。索引的创建可以使用 `CREATE INDEX` 语句。索引的维护可以使用 `VACUUM` 语句。索引的删除可以使用 `DROP INDEX` 语句。


```
CREATE INDEX idx ON featnames_short  
USING btree (upper(fullname) varchar_pattern_ops);
```

このインデックスは、`upper` 関数で `fullname` を大文字に変換し、`varchar_pattern_ops` オペレーターを使用して、`featnames_short` テーブルの `fullname` カラムに適用されています。

```
SELECT fullname FROM featnames_short WHERE upper(fullname) LIKE  
'S%';
```



このクエリは、`featnames_short` テーブルから `fullname` カラムの値が 'S%' で始まるレコードを抽出します。

例

6.3.4 インデックスの作成

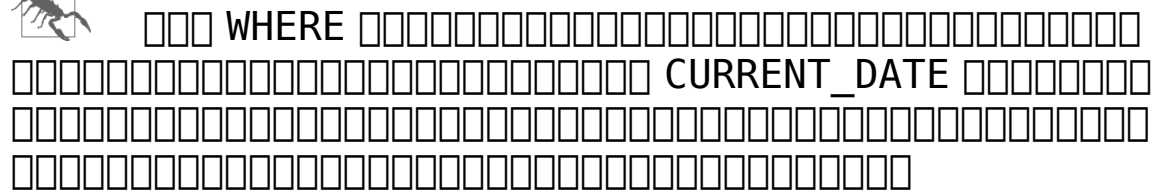
このインデックスは、`featnames_short` テーブルの `fullname` カラムに適用されています。WHERE 句で `upper(fullname) LIKE 'S%'` と指定されているため、このインデックスは、`fullname` カラムの値が 'S%' で始まるレコードを抽出するために使用されます。

このインデックスは、`featnames_short` テーブルの `fullname` カラムに適用されています。10 000 000 レコードのテーブルに対して、10 000 レコードのクエリを実行する際に、このインデックスは、`fullname` カラムの値が 'S%' で始まるレコードを抽出するために使用されます。

```
CREATE TABLE subscribers (  
  id serial PRIMARY KEY,  
  name varchar(50) NOT NULL, type varchar(50),  
  is_active boolean);'
```

このインデックスは、`subscribers` テーブルの `name` カラムに適用されています。

```
CREATE UNIQUE INDEX uq ON subscribers USING btree(lower(name))  
WHERE is_active;
```



```
CREATE OR REPLACE VIEW vw_subscribers_current AS
SELECT id, lower(name) As name FROM subscribers WHERE is_active =
true;
```

```
SELECT * FROM vw_subscribers_current WHERE name = 'sandy';
```

6.3.5 〇〇〇〇

[illegible]


```
UPDATE census.vw_facts_2011 SET val = 1 WHERE yr = 2012;
```

この更新は、2011年のデータのみを対象に、val の値を 1 に更新する。7-2 の図を参照。

図 7-2 UPDATE 文でデータを更新する

```
UPDATE census.vw_facts_2011 SET yr = 2012 WHERE yr = 2011;
```

図 7-2 の UPDATE 文は、2011 年のデータを 2012 年に更新する。9.4 節の WITH CHECK OPTION 文は、更新されたデータが元のデータの範囲内であることを確認する。vs_facts_2011 というビューは、2011 年のデータのみを対象に、yr を 2011 に更新する。7-3 の図を参照。

図 7-3 WITH CHECK OPTION 文でデータを更新する

```
CREATE OR REPLACE VIEW census.vw_facts_2011 AS  
SELECT fact_type_id, val, yr, tract_id FROM census.facts  
WHERE yr = 2011 WITH CHECK OPTION;
```

このビューは、2011 年のデータのみを対象に、val の値を 1 に更新する。

```
UPDATE census.vw_facts_2011 SET yr = 2012 WHERE val > 2942;
```

この更新は、2011 年のデータのみを対象に、yr を 2012 に更新する。

```
ERROR: new row violates WITH CHECK OPTION for view"vw_facts_2011"  
DETAIL: Failing row contains (1, 25001010500, 2012, 2985.000,  
100.00).
```

7.1.2 视图

视图是数据库中的一种对象，它是由一个或多个表中的表字段组成的。视图可以看作是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行。视图可以像表一样被查询，也可以像表一样被更新。视图可以简化复杂的查询，提高查询效率。视图还可以用于数据的安全控制，通过视图可以限制用户对数据的访问。PostgreSQL 支持视图，并且提供了丰富的视图功能。本章将介绍 PostgreSQL 视图的创建、使用和管理。

图 7-4 视图

图 7-4 视图 vw_facts

```
CREATE OR REPLACE VIEW census.vw_facts AS
SELECT
    y.fact_type_id, y.category, y.fact_subcats, y.short_name,
    x.tract_id, x.yr, x.val, x.perc
FROM census.facts As x INNER JOIN census.lu_fact_types As y
ON x.fact_type_id = y.fact_type_id;
```

视图的创建语法如下：CREATE VIEW 视图名 (列名) INSTEAD OF 表名。视图的更新语法如下：INSERT INTO 视图名 (列名) VALUES (值); UPDATE 视图名 SET 列名 = 值; DELETE FROM 视图名; TRUNCATE 视图名; SQL 语句。图 7-5 视图 PL/pgSQL 语句。

图 7-5 视图 vw_facts 的 INSERT、UPDATE、DELETE 语句

```
CREATE OR REPLACE FUNCTION census.trig_vw_facts_ins_upd_del()
RETURNS trigger AS
$$
BEGIN
    IF (TG_OP = 'DELETE') THEN ❶
        DELETE FROM census.facts AS f
        WHERE
            f.tract_id = OLD.tract_id AND f.yr = OLD.yr AND
            f.fact_type_id = OLD.fact_type_id;
        RETURN OLD;
    END IF;
    IF (TG_OP = 'INSERT') THEN ❷
```


④ OLD NEW

7-6

7-6

```
CREATE TRIGGER trig_01_vw_facts_ins_upd_del
INSTEAD OF INSERT OR UPDATE OR DELETE ON census.vw_facts
FOR EACH ROW EXECUTE PROCEDURE census.trig_vw_facts_ins_upd_del();
```

SQL

facts

```
UPDATE census.vw_facts SET yr = 2012
WHERE yr = 2011 AND tract_id = '25027761200';
```

PostgreSQL

Query returned successfully: 56 rows affected, 40 ms execution time.

```
UPDATE census.vw_facts SET short_name = 'test';
```

Query returned successfully: 0 rows affected, 931 ms execution time.

INSERT UPDATE DELETE

PostgreSQL rules INSTEAD OF “Database Abstraction with Updateable Views” rules

rules INSTEAD OF PostgreSQL rules INSTEAD OF SELECT rules SQL rules rules SQL

7.1.3

REFRESH MATERIALIZED VIEW 9.3

OLAP

7-7 7-1

7-7

```
CREATE MATERIALIZED VIEW census.vw_facts_2011_materialized AS
SELECT fact_type_id, val, yr, tract_id FROM census.facts WHERE yr =
2011;
```

7-8

7-8

```
CREATE UNIQUE INDEX ix
ON census.vw_facts_2011_materialized (tract_id, fact_type_id, yr);
```


[illegible]

- `REFRESH MATERIALIZED VIEW` `PostgreSQL` `crontab` `pgAgent` “Caching Data-with Materialized Views and Statement-Level- Trigger”
- `9.3` `CONCURRENTLY` `REFRESH`

7.2 PostgreSQLSQL

PostgreSQL SQL 数据库 PostgreSQL 数据库
 PostgreSQL SQL 数据库 PostgreSQL 数据库“”
 ANSI SQL 数据库 ANSI SQL 数据库
 数据库

7.2.1 DISTINCT ON

```

DISTINCT ON (DISTINCT
DISTINCT ON
DISTINCT ON
DISTINCT ON
ON

```

7-10

7-10 DISTINCT ON

```
SELECT DISTINCT ON (left(tract_id, 5))
    left(tract_id, 5) As county, tract_id, tract_name
FROM census.lu_tracts
ORDER BY county, tract_id;
```

```

county | tract_id | tract_name
-----+-----+-----
-----

```

```

25001 | 25001010100 | Census Tract 101, Barnstable County,
Massachusetts
25003 | 25003900100 | Census Tract 9001, Berkshire County,
Massachusetts
25005 | 25005600100 | Census Tract 6001, Bristol County,
Massachusetts
25007 | 25007200100 | Census Tract 2001, Dukes County,
Massachusetts
25009 | 25009201100 | Census Tract 2011, Essex County,
Massachusetts
:
(14 rows)

```

ORDER BY clause sorts the results of the query. The **DISTINCT ON** clause is used to return only one row for each group of rows that are identical in the columns specified in the **DISTINCT ON** clause.

7.2.2 LIMIT and OFFSET

LIMIT and **OFFSET** are used to limit the number of rows returned by a query. **ORDER BY** is used to sort the results of the query. The **7-11** example shows how to use **LIMIT** and **OFFSET** to return the first 3 rows of the results, skipping the first 0 rows.

PostgreSQL and MySQL both support the **LIMIT** and **OFFSET** clauses.

7-11 Example 7-10: Using **LIMIT** and **OFFSET** to return the first 3 rows of the results, skipping the first 0 rows.

```

SELECT DISTINCT ON (left(tract_id, 5))
    left(tract_id, 5) As county, tract_id, tract_name
FROM census.lu_tracts
ORDER BY county, tract_id LIMIT 3 OFFSET 2;

county | tract_id      | tract_name
-----+-----+-----
----
25005  | 25005600100  | Census Tract 6001, Bristol County,
Massachusetts
25007  | 25007200100  | Census Tract 2001, Dukes County,
Massachusetts
25009  | 25009201100  | Census Tract 2011, Essex County,

```

Massachusetts
(3 rows)

7.2.3 轉換日期格式

ANSI SQL 使用 CAST 轉換日期格式
CAST('2011-1-1' AS date) 轉換 2011-1-1 日期格式
PostgreSQL 使用 ::date 轉換日期格式
'2011-1-1'::date
A B C
someXML::text::integer

7.2.4 插入數據

PostgreSQL 7-12 6-3

7-12

```
INSERT INTO logs_2011 (user_name, description, log_ts)
VALUES
    ('robe', 'logged in', '2011-01-10 10:15 AM EST'),
    ('lhsu', 'logged out', '2011-01-11 10:20 AM EST');
```

PostgreSQL 使用 VALUES 插入數據
7-13

7-13 VALUES

```
SELECT *
FROM (
    VALUES
        ('robe', 'logged in', '2011-01-10 10:15 AM
EST'::timestamp),
        ('lhsu', 'logged out', '2011-01-11 10:20 AM
EST'::timestamp)
) AS l (user_name, description, log_ts);
```

VALUES 表の列の値を指定する。MySQL の SQL Server と異なる。

7.2.5 ILIKE 演算子

PostgreSQL では、ANSI LIKE 演算子と upper 関数を用いて、大文字小文字を区別せずに検索できる。PostgreSQL では、ILIKE 演算子も利用できる。

```
SELECT tract_name FROM census.lu_tracts WHERE tract_name ILIKE
'%duke%';
tract_name
-----
Census Tract 2001, Dukes County, Massachusetts
Census Tract 2002, Dukes County, Massachusetts
Census Tract 2003, Dukes County, Massachusetts
Census Tract 2004, Dukes County, Massachusetts
Census Tract 9900, Dukes County, Massachusetts
```

7.2.6 ANY 演算子

PostgreSQL では、ANY 演算子を用いて、複数の値の中から任意の値を検索できる。ANY 演算子は、配列の要素に対して使用される。

例

```
SELECT tract_name
FROM census.lu_tracts
WHERE tract_name ILIKE
ANY(ARRAY['%99%duke%', '%06%Barnstable%']::text[]);
tract_name
-----
Census Tract 102.06, Barnstable County, Massachusetts
Census Tract 103.06, Barnstable County, Massachusetts
Census Tract 106, Barnstable County, Massachusetts
Census Tract 9900, Dukes County, Massachusetts
(4 rows)
```

이러한 경우 `ILIKE` 또는 `OR` 연산자를 사용하여
어떤 문자열이 `ANY` 문자열과 `LIKE` = ~ 문자열
문자열 `like` 문자열과 일치하는지

`ANY` 문자열과 일치하는지 확인하는 방법은
문자열과 일치하는지 확인하는 방법은
문자열과 일치하는지 확인하는 방법은

7.2.7 문자열 일치

문자열 일치

PostgreSQL에서 `SELECT` 문을 사용하여
문자열 일치

문자열 SQL 문을 사용하여
문자열 일치
문 7-14 문자열 `generate_series` 함수

```
CREATE TABLE interval_periods (i_type interval);
INSERT INTO interval_periods (i_type)
VALUES ('5 months'), ('132 days'), ('4862 hours');
```

문 7-14 `SELECT` 문을 사용하여

```
SELECT i_type,
       generate_series('2012-01-01'::date, '2012-12-31'::date, i_type)
As dt
FROM interval_periods;
```

| i_type | dt |
|------------|------------------------|
| 5 months | 2012-01-01 00:00:00-05 |
| 5 months | 2012-06-01 00:00:00-04 |
| 5 months | 2012-11-01 00:00:00-04 |
| 132 days | 2012-01-01 00:00:00-05 |
| 132 days | 2012-05-12 00:00:00-04 |
| 132 days | 2012-09-21 00:00:00-04 |
| 4862 hours | 2012-01-01 00:00:00-05 |

7.2.8 DELETE UPDATE INSERT

DELETE UPDATE INSERT

PostgreSQL ONLY 7-37 ONLY

7.2.9 DELETE USING

“” USING WHERE USING FROM USING 7-15 census.facts short_name='s01'

7-15 DELETE USING

```
DELETE FROM census.facts
USING census.lu_fact_types As ft
WHERE facts.fact_type_id = ft.fact_type_id AND ft.short_name =
's01';
```

WHERE IN

7.2.10

RETURNING ANSI SQL 7-37 RETURNING DELETE INSERT UPDATE RETURNING serial RETURNING serial serial

RETURNING serial
RETURNING * 7-16

7-16 UPDATE RETURNING

```
UPDATE census.lu_fact_types AS f
SET short_name = replace(replace(lower(f.fact_subcats[4]),
',','_'),':','')
WHERE f.fact_subcats[3] = 'Hispanic or Latino:' AND
f.fact_subcats[4] > ''
RETURNING fact_type_id, short_name;
```

| fact_type_id | short_name |
|--------------|--|
| 96 | white_alone |
| 97 | black_or_african_american_alone |
| 98 | american_indian_and_alaska_native_alone |
| 99 | asian_alone |
| 100 | native_hawaiian_and_other_pacific_islander_alone |
| 101 | some_other_race_alone |
| 102 | two_or_more_races |

7.2.11 UPSERT INSERT UPDATE

PostgreSQL 9.5 INSERT ON CONFLICT UPSERT

```
CREATE TABLE colors(color varchar(50) PRIMARY KEY, hex varchar(6));
INSERT INTO colors(color, hex)
VALUES('blue', '0000FF'), ('red', 'FF0000');
```

7-17 UPSERT

green SQL

7-17

```
INSERT INTO colors(color, hex)
VALUES('blue', '0000FF'), ('red', 'FF0000'), ('green',
'00FF00')
ON CONFLICT DO NOTHING ;
```

“Blue” “blue”
“”

```
CREATE UNIQUE INDEX uidx_colors_lcolor ON colors USING
btree(lower(color));
```

“Blue” ON CONFLICT DO
NOTHING “Blue”
“blue” 7-18

7-18 ON CONFLICT DO UPDATE

```
INSERT INTO colors(color, hex)
VALUES('Blue', '0000FF'), ('Red', 'FF0000'), ('Green', '00FF00')
ON CONFLICT(lower(color))
DO UPDATE SET color = EXCLUDED.color, hex = EXCLUDED.hex;
```

7-18 ON CONFLICT lower(color)
upper(color)
colors upser(color)

ON CONFLICT ON CONFLICT
ON CONFLICT ON CONSTRAINT
+ 7-19

7-19 ON CONFLICT DO UPDATE

```
INSERT INTO colors(color, hex)
VALUES('Blue', '0000FF'), ('Red', 'FF0000'), ('Green',
'00FF00')
ON CONFLICT ON CONSTRAINT colors_pkey
DO UPDATE SET color = EXCLUDED.color, hex = EXCLUDED.hex;
```

```

00000000000000000000000000000000D0 00000000000000000000000000000000
00000000000000000000000000000000D0 UPDATE 00000000

```

7.2.12

```
SELECT x FROM census.lu fact types As x LIMIT 2;
```

[illegible]

```
x
-----
(86,Population,"{D001,Total:}",d001)
(87,Population,"{D002,Total:,""Not Hispanic or Latino:""}",d002)
```

```

lu_fact_type  Population  category  {D001,Total:}  fact_subcats  array_agg  hstore
hstore  hstore

```

Web PostgreSQL JSON JSONB
JSON JSONB 5.6 array_agg
array_to_json JSON
7-20

例 7-20 array_agg 関数で JSON 形式

```
SELECT array_to_json(array_agg(f)) As cat ❶
FROM (
    SELECT MAX(fact_type_id) As max_type, category ❷
    FROM census.lu_fact_types
    GROUP BY category
) As f;
```

実行結果

```
cats
-----
[{"max_type":102,"category":"Population"},
{"max_type":153,"category":"Housing"}]
```

❶ array_agg 関数で f を配列として集約

❷ array_agg 関数で集約された配列を array_to_json 関数で JSON 形式に変換

PostgreSQL 9.3 から json_agg 関数が追加された。array_to_json 関数と array_agg 関数を組み合わせて、図 7-21 のように json_agg 関数で 7-20 の実行結果を再現できる。

例 7-21 json_agg 関数で JSON 形式

```
SELECT json_agg(f) As cats
FROM (
    SELECT MAX(fact_type_id) As max_type, category
    FROM census.lu_fact_types
    GROUP BY category
) As f;
```

7.2.13 \$\$ を使う

ANSI SQL では、文字列リテラル ' ' を使って、文字列を定義する。ただし、文字列中に ' ' が含まれる場合は、' ' をエスケープする必要がある。例えば、' ' を含む文字列を定義するには、' ' を ' ' と書く必要がある。

이제 O'Nan은 mon's place에서
can't...
"..." INSERT ...
...
...

PostgreSQL에서 \$\$...
...

\$... SQL ... exec(...) ... 7-5 ...
\$\$...

SQL ... ANSI ...
...

```
SELECT 'It's O'Neil's play. ' || 'It'll start at two o'clock.'
```

\$\$...

```
SELECT $$It's O'Neil's play. $$ || $$It'll start at two o'clock.$$
```

...

\$\$...

\$...

7.2.14 DO

DO ...
... 3-10 ...
PL/pgSQL ...

...

```

set search_path=census;
DROP TABLE IF EXISTS lu_fact_types CASCADE;
CREATE TABLE lu_fact_types (
    fact_type_id serial,
    category varchar(100),
    fact_subcats varchar(255)[],
    short_name varchar(50),
    CONSTRAINT pk_lu_fact_types PRIMARY KEY (fact_type_id)
);

```

Figure D0 illustrates the process of dropping the table `lu_fact_types` and creating it again with the `CASCADE` option. The `CASCADE` option ensures that the table is dropped even if it has dependent objects.

Figure 7-22 shows the `INSERT INTO SELECT` statement used to populate the `lu_fact_types` table. The SQL statement is as follows:



Figure 7-22 illustrates the process of populating the `lu_fact_types` table using the `INSERT INTO SELECT` statement. The SQL statement is as follows:

Figure 7-22: Figure D0 illustrates the process of dropping the table `lu_fact_types` and creating it again with the `CASCADE` option. The `CASCADE` option ensures that the table is dropped even if it has dependent objects.

```

D0 language plpgsql
$$
DECLARE var_sql text;
BEGIN
    var_sql := string_agg(
        $sql$ ❶
        INSERT INTO lu_fact_types(category, fact_subcats,
short_name)
        SELECT
            'Housing',
            array_agg(s$sql$ || lpad(i::text,2,'0')
                || ') As fact_subcats,'
                || quote_literal('s' || lpad(i::text,2,'0')) || ' As
short_name
            FROM staging.factfinder_import
            WHERE s' || lpad(I::text,2,'0') || $sql$ ~ '^[a-zA-Z]+'
        $sql$, ';'
    );

```


7-24 FILTER 7-24

7-24 AVG FILTER

```
SELECT student,  
       AVG(score) FILTER (WHERE subject = 'algebra') As algebra,  
       AVG(score) FILTER (WHERE subject = 'physics') As physics  
FROM test_scores  
GROUP BY student;
```

CASE FILTER
FILTER CASE
NULL array_agg NULL
CASE WHEN 7-25
CASE...WHEN...
[]

7-25 CASE WHEN array_agg

```
SELECT student,  
       array_agg(CASE WHEN subject = 'algebra' THEN score ELSE NULL  
END) As algebra,  
       array_agg(CASE WHEN subject = 'physics' THEN score ELSE NULL  
END) As physics  
FROM test_scores  
GROUP BY student;
```

| student | algebra | physics |
|---------|---------------------------|-------------------------------|
| jojo | {74,NULL,NULL,NULL,74,..} | {NULL,72,NULL,NULL,NULL,72..} |
| jdoe | {75,NULL,NULL,NULL,78,..} | {NULL,72,NULL,NULL,NULL,72..} |
| robe | {68,NULL,NULL,NULL,77,..} | {NULL,72,NULL,NULL,NULL,72..} |
| lhsu | {84,NULL,NULL,NULL,80,..} | {NULL,72,NULL,NULL,NULL,72..} |

(4 rows)

Figure 7-25 illustrates how NULL values are handled in the FILTER clause. Figure 7-26 illustrates how the FILTER clause is used to filter out NULL values.

Figure 7-26 FILTER with array_agg

```
SELECT student,
       array_agg(score) FILTER (WHERE subject = 'algebra') As algebra,
       array_agg(score) FILTER (WHERE subject = 'physics') As physics
FROM test_scores
GROUP BY student;
```

| student | algebra | physics |
|---------|---------|---------|
| jojo | {74,74} | {83,79} |
| jdoe | {75,78} | {72,72} |
| robe | {68,77} | {83,85} |
| lhsu | {84,80} | {72,72} |

FILTER is supported in PostgreSQL 9.4 and later. It is not supported in earlier versions of PostgreSQL.

7.2.16 Percentile Functions

PostgreSQL 9.4 introduced two new aggregate functions: `percentile_disc` and `percentile_cont`. These functions return the value at the specified percentile. The `mode` function returns the most frequently occurring value in a dataset.

`percentile_disc` and `percentile_cont` are used to find the value at a specific percentile. For example, `percentile_disc(0.5)` returns the median value. `percentile_cont(0.5)` returns the value at the 50th percentile.

Figure 7-27 illustrates how the `mode` function is used to find the most frequently occurring value in a dataset.

Figure 7-27 mode

```
SELECT
  student,
```

```

        percentile_cont(0.5) WITHIN GROUP (ORDER BY score) As
cont_median,
        percentile_disc(0.5) WITHIN GROUP (ORDER BY score) AS
disc_median,
        mode() WITHIN GROUP (ORDER BY score) AS mode,
        COUNT(*) As num_scores
FROM test_scores
GROUP BY student
ORDER BY student;

```

| student | cont_median | disc_median | mode | num_scores |
|---------|-------------|-------------|------|------------|
| alex | 78 | 77 | 74 | 8 |
| leo | 72 | 72 | 72 | 8 |
| regina | 76 | 76 | 68 | 9 |
| sonia | 73.5 | 72 | 72 | 8 |

(4 rows)

例 7-27 百分位和众数

百分位和众数都是统计函数，它们都使用 WITHIN GROUP 子句 ORDER BY 子句。

百分位和众数都是统计函数，它们都使用 WITHIN GROUP 子句 ORDER BY 子句。

例 7-28 百分位和众数

```

SELECT
    student,
    percentile_cont('{0.5,0.60,1}':float[])
WITHIN GROUP (ORDER BY score) AS cont_median,
    percentile_disc('{0.5,0.60,1}':float[])
WITHIN GROUP (ORDER BY score) AS disc_median,
    COUNT(*) As num_scores
FROM test_scores
GROUP BY student
ORDER BY student;

```

| student | cont_median | disc_median | num_scores |
|---------|-------------|-------------|------------|
|---------|-------------|-------------|------------|

| | | | |
|--------|----------------|------------|---|
| alex | {78,79.2,84} | {77,79,84} | 8 |
| leo | {72,73.6,84} | {72,72,84} | 8 |
| regina | {76,76.8,90} | {76,77,90} | 9 |
| sonia | {73.5,75.6,86} | {72,75,86} | 8 |

(4 rows)

7-29 `WITHIN GROUP` `FILTER`

7-29

```

SELECT
    student,
    percentile_disc(0.5) WITHIN GROUP (ORDER BY score)
        FILTER (WHERE subject = 'algebra') AS algebra,
    percentile_disc(0.5) WITHIN GROUP (ORDER BY score)
        FILTER (WHERE subject = 'physics') AS physics
FROM test_scores
GROUP BY student
ORDER BY student;

```

| student | algebra | physics |
|---------|---------|---------|
| alex | 74 | 79 |
| leo | 80 | 72 |
| regina | 68 | 83 |
| sonia | 75 | 72 |

(4 rows)

7.3

ANSI SQL `row_number` `rank`

SQL

OVER 子句は、OVER 子句で指定された範囲内で、AVERAGE 関数を使用して、fact_type_id=86 の val の平均値を計算します。PostgreSQL では、AVG 関数を使用して、GROUP BY 子句で指定された範囲内で、JOIN 子句で指定された範囲内で、

図 7-30 図 7-30 は、SELECT 文を使用して、fact_type_id=86 の val の平均値を計算する SQL 文を示しています。WHERE 子句で指定された範囲内で、JOIN 子句で指定された範囲内で、

図 7-30 図 7-30 は、SELECT 文を使用して、fact_type_id=86 の val の平均値を計算する SQL 文を示しています。

```
SELECT tract_id, val, AVG(val) OVER () as val_avg
FROM census.facts
WHERE fact_type_id = 86;
```

| tract_id | val | val_avg |
|-------------|----------|-----------------------|
| 25001010100 | 2942.000 | 4430.0602165087956698 |
| 25001010206 | 2750.000 | 4430.0602165087956698 |
| 25001010208 | 2003.000 | 4430.0602165087956698 |
| 25001010304 | 2421.000 | 4430.0602165087956698 |
| : | | |

OVER 子句は、OVER 子句で指定された範囲内で、AVERAGE 関数を使用して、fact_type_id=86 の val の平均値を計算します。PostgreSQL では、AVG 関数を使用して、GROUP BY 子句で指定された範囲内で、JOIN 子句で指定された範囲内で、

SQL 文は、OVER 子句で指定された範囲内で、ROW RANK LEAD 関数を使用して、PostgreSQL では、AVG 関数を使用して、GROUP BY 子句で指定された範囲内で、JOIN 子句で指定された範囲内で、

7.3.1 PARTITION BY

OVER 子句は、OVER 子句で指定された範囲内で、PARTITION BY 子句を使用して、PostgreSQL では、AVG 関数を使用して、GROUP BY 子句で指定された範囲内で、JOIN 子句で指定された範囲内で、

Figure 7-31 shows the results of the query. The results are ordered by tract_id, and the average value for each tract is calculated.

Figure 7-31 Results of the query for Figure 7-30

```
SELECT tract_id, val, AVG(val) OVER (PARTITION BY left(tract_id,5))
As val_avg_county
FROM census.facts
WHERE fact_type_id = 2 ORDER BY tract_id;
```

| tract_id | val | val_avg_county |
|-------------|----------|-----------------------|
| 25001010100 | 1765.000 | 1709.9107142857142857 |
| 25001010206 | 1366.000 | 1709.9107142857142857 |
| 25001010208 | 984.000 | 1709.9107142857142857 |
| : | | |
| 25003900100 | 1920.000 | 1438.2307692307692308 |
| 25003900200 | 1968.000 | 1438.2307692307692308 |
| 25003900300 | 1211.000 | 1438.2307692307692308 |

7.3.2 ORDER BY

The `ORDER BY` clause is used to sort the results of a query. The `ORDER BY` clause can be used to sort the results of a query by a single column or by multiple columns. The `ORDER BY` clause can also be used to sort the results of a query in ascending or descending order. The `ROW_NUMBER` function is used to assign a unique number to each row in a result set. Figure 7-32 shows the results of the query.

Figure 7-32 Results of the query for Figure 7-31

```
SELECT ROW_NUMBER() OVER (ORDER BY tract_name) As rnum, tract_name
FROM census.lu_tracts
ORDER BY rnum LIMIT 4;
```

| rnum | tract_name |
|------|--|
| 1 | Census Tract 1, Suffolk County, Massachusetts |
| 2 | Census Tract 1001, Suffolk County, Massachusetts |
| 3 | Census Tract 1002, Suffolk County, Massachusetts |
| 4 | Census Tract 1003, Suffolk County, Massachusetts |

例 7-32 ORDER BY 関数 OVER 関数を使用したクエリ

PARTITION BY 句 ORDER BY 句を使用したクエリ PARTITION BY 句
例 7-33 ORDER BY 句 OVER 関数 PARTITION BY
句 ORDER BY 句

例 7-33 PARTITION BY 句 ORDER BY

```
SELECT tract_id, val,  
       SUM(val) OVER (PARTITION BY left(tract_id,5) ORDER BY val) As  
sum_county_ordered  
FROM census.facts  
WHERE fact_type_id = 2  
ORDER BY left(tract_id,5), val;
```

| tract_id | val | sum_county_ordered |
|-------------|---------|--------------------|
| 25001014100 | 226.000 | 226.000 |
| 25001011700 | 971.000 | 1197.000 |
| 25001010208 | 984.000 | 2181.000 |
| : | | |
| 25003933200 | 564.000 | 564.000 |
| 25003934200 | 593.000 | 1157.000 |
| 25003931300 | 606.000 | 1763.000 |

例 7-33 ORDER BY 句 OVER 関数 PARTITION BY 句
ORDER BY 句を使用したクエリ ORDER BY 句
PARTITION BY 句 ORDER BY 句
left(tract_id,5), val 句 OVER 関数 PARTITION BY
句 ORDER BY 句 ORDER BY 句

RANGE 句 ROWS 句 BETWEEN
CURRENT ROW AND 5 FOLLOWING 句

PostgreSQL 関数 LEAD 関数 LAG 関数
例 7-34 LEAD 関数 LAG 関数
関数

7-34 LEAD LAG

```
SELECT * FROM (
    SELECT
        ROW_NUMBER() OVER( wt ) As rnum, ❶
        substring(tract_id,1, 5) As county_code,
        tract_id,
        LAG(tract_id,2) OVER wt As tract_2_before,
        LEAD(tract_id) OVER wt As tract_after
    FROM census.lu_tracts
    WINDOW wt AS (PARTITION BY substring(tract_id,1, 5) ORDER BY
tract_id) ❷
) As x
WHERE rnum BETWEEN 2 and 3 AND county_code IN ('25007','25025')
ORDER BY county_code, rnum;
```

| rnum | county_code | tract_id | tract_2_before | tract_after |
|------|-------------|-------------|----------------|-------------|
| 2 | 25007 | 25007200200 | | 25007200300 |
| 3 | 25007 | 25007200300 | 25007200100 | 25007200400 |
| 2 | 25025 | 25025000201 | | 25025000202 |
| 3 | 25025 | 25025000202 | 25025000100 | 25025000301 |

[illegible]

② □□□□□□□□ wt □□□

```
LEAD | LAG | step | ... | LEAD | LAG | ... | NULL
```

[illegible]

7.4 CTE□□□

CTE SQL
SQL CTE
CTE

CTE

CTE

CTE SQL CTE
SQL CTE

CTE

CTE UPDATE INSERT DELETE
CTE

CTE

CTE CTE CTE

PostgreSQL CTE

7.4.1 CTE

CTE 7-35 WITH CTE

7-35 CTE

```
WITH cte AS (  
    SELECT  
        tract_id, substring(tract_id,1, 5) As county_code,  
        COUNT(*) OVER(PARTITION BY substring(tract_id,1, 5)) As  
cnt_tracts  
    FROM census.lu_tracts  
)  
SELECT MAX(tract_id) As last_tract, county_code, cnt_tracts  
FROM cte  
WHERE cnt_tracts > 100  
GROUP BY county_code, cnt_tracts;
```

例 7-35 例 CTE 例例例例 cte 例例例例例 SELECT 例例例例例例例例例
例例例 tract_id 例country_code 例cnt_tracts 例例例例例 SQL 例
例例例 CTE 例例例例例例例例例

例 SQL 例例例例例例例 CTE例CTE 例例例例例例例例例 CTE 例例例例例
WITH 例例例例例例例例例例 7-36 例例例 CTE 例例例例例例例例例例例例例
例 CTE 例例例例例例例 CTE例例例例例例例例例例例例例 CTE 例例例例例例例例例
例例例

例 7-36 例 CTE 例例例

```
WITH
    cte1 AS(
        SELECT
            tract_id,
            substring(tract_id,1, 5) As county_code,
            COUNT(*) OVER (PARTITION BY substring(tract_id,1,5)) As
cnt_tracts
        FROM census.lu_tracts
    ),
    cte2 AS (
        SELECT
            MAX(tract_id) As last_tract,
            county_code,
            cnt_tracts
        FROM cte1
        WHERE cnt_tracts < 8 GROUP BY county_code, cnt_tracts
    )
SELECT c.last_tract, f.fact_type_id, f.val
FROM census.facts As f INNER JOIN cte2 c ON f.tract_id =
c.last_tract;
```

7.4.2 例 CTE例例例

例 CTE 例例 CTE 例例例例例例例例例例例例例例例 6-3 例例例例例例例例例
例例例例例例例例例

```
CREATE TABLE logs_2011_01_02 (
    PRIMARY KEY (log_id),
    CONSTRAINT chk
        CHECK (log_ts >= '2011-01-01' AND log_ts < '2011-03-01')
```

```
)
INHERITS (logs_2011);
```

図 7-37 2011 年 1 月 2 日以前にのみデータを返す SQL 文の例。RETURNING 句は 7.2.10 節を参照。

図 7-37 CTE を用いた SQL 文の例

```
WITH t AS (
    DELETE FROM ONLY logs_2011 WHERE log_ts < '2011-03-01'
    RETURNING *
)
INSERT INTO logs_2011_01_02 SELECT * FROM t;
```

7.4.3 CTE の使用

PostgreSQL では CTE を用いて「**RECURSIVE**」という CTE を用いて SQL 文を記述する。CTE は UNION ALL を用いて結合される。

CTE は WITH 句で定義され、RECURSIVE 句は WITH 句の後に記述される。RECURSIVE 句は CTE を用いて結合される。

CTE は「Recursive CTE to Display Tree Structures」で説明されている。

図 7-38 catalog テーブルの例

図 7-38 CTE

```
WITH RECURSIVE tbls AS (
    SELECT
        c.oid As tableoid,
        n.nspname AS schemaname,
        c.relname AS tablename ❶
```



```

FROM
    pg_class c LEFT JOIN
    pg_namespace n ON n.oid = c.relnamespace LEFT JOIN
    pg_tablespace t ON t.oid = c.reltablespace LEFT JOIN
    pg_inherits As th ON th.inhrelid = c.oid
WHERE
    th.inhrelid IS NULL AND
    c.relkind = 'r'::"char" AND c.relhassubclass
UNION ALL
SELECT
    c.oid As tableoid,
    n.nspname AS schemaname,
    tbls.tablename || '->' || c.relname AS tablename ❷ ❸
FROM
    tbls INNER JOIN
    pg_inherits As th ON th.inhparent = tbls.tableoid INNER
JOIN
    pg_class c ON th.inhrelid = c.oid LEFT JOIN
    pg_namespace n ON n.oid = c.relnamespace LEFT JOIN
    pg_tablespace t ON t.oid = c.reltablespace
)
SELECT * FROM tbls ORDER BY tablename; ❹

```

| tableoid | schemaname | tablename |
|----------|------------|----------------------------------|
| 3152249 | public | logs |
| 3152260 | public | logs->logs_2011 |
| 3152272 | public | logs->logs_2011->logs_2011_01_02 |

❶ 継承関係のあるクラスを除外

❷ 継承関係のあるクラスを除外し、tbls テーブルに結合

❸ 継承関係のあるクラスを除外し、tbls テーブルに結合

❹ 継承関係のあるクラスを除外し、tbls テーブルに結合し、tbls テーブルに結合

7.5 LATERAL 結合

LATERAL 結合 9.3 節参照 ANSI SQL 標準に準拠した結合方法

□□□□□□ L.year=2011 □□□□□□□□□□□□□□□□

```
SELECT *
  FROM
    census.facts L
  INNER JOIN
    (
      SELECT *
      FROM census.lu_fact_types
      WHERE category = CASE WHEN L.yr = 2011
      THEN 'Housing' ELSE category END
    ) R
  ON L.fact_type_id = R.fact_type_id;
```

000 LATERAL 000000000000

```
SELECT *
FROM census.facts L INNER JOIN LATERAL
(
    SELECT *
    FROM census.lu_fact_types
    WHERE category = CASE WHEN L.yr = 2011
THEN 'Housing' ELSE category END
) R
ON L.fact_type_id = R.fact_type_id;
```

0000 LATERAL 00000000 FROM 000000000000000000000000
0000000000000000000000000000

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX LATERAL XXXXXX 7-39 XXXXXXXX
XXXXXXXXXXXXXXXX generate series XXXXXXXX

```

```
CREATE TABLE interval_periods(i_type interval);
INSERT INTO interval_periods (i_type)
VALUES ('5 months'), ('132 days'), ('4862 hours');
```

7-39 LATERAL generate series

```

SELECT i_type, dt
FROM
    interval_periods CROSS JOIN LATERAL
    generate_series('2012-01-01'::date, '2012-12-31'::date, i_type)
AS dt
WHERE NOT (dt = '2012-01-01' AND i_type = '132 days'::interval);

```

| i_type | dt |
|------------|------------------------|
| 5 mons | 2012-01-01 00:00:00-05 |
| 5 mons | 2012-06-01 00:00:00-04 |
| 5 mons | 2012-11-01 00:00:00-04 |
| 132 days | 2012-05-12 00:00:00-04 |
| 132 days | 2012-09-21 00:00:00-04 |
| 4862:00:00 | 2012-01-01 00:00:00-05 |
| 4862:00:00 | 2012-07-21 15:00:00-04 |

LATERAL 関数は、クエリで参照されるテーブルのサブクエリを、クエリごとに実行します。7-40 の例のように、LATERAL 関数は、100 行のデータを返す <http://www.postgresonline.com> のデータベースの 5 行のデータを返すクエリを、6.1.5 から 6.1.1 までサポートしています。

7-40 LATERAL 関数の使用例

```

SELECT u.user_name, l.description, l.log_ts
FROM
    super_users AS u CROSS JOIN LATERAL (
        SELECT description, log_ts
        FROM logs
        WHERE
            log_ts > CURRENT_TIMESTAMP - interval '100 days' AND
            logs.user_name = u.user_name
        ORDER BY log_ts DESC LIMIT 5
    ) AS l;

```

LATERAL 関数は、クエリで参照されるテーブルのサブクエリを、クエリごとに実行します。

SQL クエリで LATERAL 関数を使用する場合は、LATERAL 関数の名前を LATERAL と指定する必要があります。LATERAL 関数の名前を LATERAL と指定しない場合は、クエリで参照されるテーブルのサブクエリを、クエリごとに実行しません。

■ LATERAL 是 PostgreSQL 特有的 SQL 子句。ANSI SQL 标准中没有 LATERAL 子句。

ANSI SQL 标准中有 Oracle 的 CROSS APPLY 和 OUTER APPLY 子句。

7.6 WITH ORDINALITY

PostgreSQL 9.4 引入了 ANSI SQL 的 WITH ORDINALITY 子句。WITH ORDINALITY 子句用于返回查询结果的行号。



WITH ORDINALITY 子句用于返回查询结果的行号。ROW_NUMBER 函数用于返回行号。

WITH ORDINALITY 子句与 generate_series 函数结合使用，可以生成一个包含行号的序列。WITH ORDINALITY 子句返回的序列可以用于生成行号。

图 7-41 显示了 WITH ORDINALITY 子句与 generate_series 函数的结合使用。

图 7-41 显示了 WITH ORDINALITY 子句与 generate_series 函数的结合使用。

```
SELECT dt.*
FROM generate_series('2016-01-01'::date, '2016-12-31'::date, interval
'1 month')
WITH ORDINALITY As dt;

dt                                | ordinality
-----+-----
2016-01-01 00:00:00-05           |          1
2016-02-01 00:00:00-05           |          2
2016-03-01 00:00:00-05           |          3
2016-04-01 00:00:00-04           |          4
2016-05-01 00:00:00-04           |          5
2016-06-01 00:00:00-04           |          6
2016-07-01 00:00:00-04           |          7
2016-08-01 00:00:00-04           |          8
```

| | |
|------------------------|----|
| 2016-09-01 00:00:00-04 | 9 |
| 2016-10-01 00:00:00-04 | 10 |
| 2016-11-01 00:00:00-04 | 11 |
| 2016-12-01 00:00:00-05 | 12 |

(12 rows)

WITH ORDINALITY 是表函数中 ordinality 选项 WITH ORDINALITY 是 SQL 中 FROM 子句中 ordinality 选项

WITH ORDINALITY 子句 LATERAL 子句 7-42 子句 7-39 子句 LATERAL 子句

7-42 WITH ORDINALITY LATERAL

```
SELECT d.ord, i_type, d.dt
FROM
    interval_periods CROSS JOIN LATERAL
    generate_series('2012-01-01'::date, '2012-12-31'::date, i_type)
WITH ORDINALITY AS d(dt,ord)
WHERE NOT (dt = '2012-01-01' AND i_type = '132 days'::interval);
```

| ord | i_type | dt |
|-----|------------|------------------------|
| 1 | 5 mons | 2012-01-01 00:00:00-05 |
| 2 | 5 mons | 2012-06-01 00:00:00-04 |
| 3 | 5 mons | 2012-11-01 00:00:00-04 |
| 2 | 132 days | 2012-05-12 00:00:00-04 |
| 3 | 132 days | 2012-09-21 00:00:00-04 |
| 1 | 4862:00:00 | 2012-01-01 00:00:00-05 |
| 2 | 4862:00:00 | 2012-07-21 15:00:00-04 |

(7 rows)

7-42 WITH ORDINALITY 子句 WHERE 子句 FROM 子句 132 days 子句 1 子句 WHERE 子句

7.7 GROUPING SETS CUBE ROLLUP

GROUPING SETS 子句

Figure 7-43 SQL query using GROUPING SETS

Figure 7-43 SQL query using GROUPING SETS

```
SELECT student, subject, AVG(score)::numeric(10,2)
FROM test_scores
WHERE student IN ('leo','regina')
GROUP BY GROUPING SETS ((student),(student,subject))
ORDER BY student, subject NULLS LAST;
```

| student | subject | avg |
|---------|-----------|-------|
| leo | algebra | 82.00 |
| leo | calculus | 65.50 |
| leo | chemistry | 75.50 |
| leo | physics | 72.00 |
| leo | NULL | 73.75 |
| regina | algebra | 72.50 |
| regina | calculus | 64.50 |
| regina | chemistry | 73.50 |
| regina | economics | 90.00 |
| regina | physics | 84.00 |
| regina | NULL | 75.44 |

(11 rows)

Figure 7-43 SQL query using GROUPING SETS

Figure 7-44 SQL query using GROUPING SETS

Figure 7-44 SQL query using GROUPING SETS

```
SELECT student, subject, AVG(score)::numeric(10,2)
FROM test_scores
WHERE student IN ('leo','regina')
GROUP BY GROUPING SETS ((student,subject),(student),(subject))
ORDER BY student NULLS LAST, subject NULLS LAST;
```

| student | subject | avg |
|---------|----------|-------|
| leo | algebra | 82.00 |
| leo | calculus | 65.50 |

| | | |
|--------|-----------|-------|
| leo | chemistry | 75.50 |
| leo | physics | 72.00 |
| leo | NULL | 73.75 |
| regina | algebra | 72.50 |
| regina | calculus | 64.50 |
| regina | chemistry | 73.50 |
| regina | economics | 90.00 |
| regina | physics | 84.00 |
| regina | NULL | 75.44 |
| NULL | algebra | 77.25 |
| NULL | calculus | 65.00 |
| NULL | chemistry | 74.50 |
| NULL | economics | 90.00 |
| NULL | physics | 78.00 |

(16 rows)

GROUPING SETS((student),(student, subject),
 ()) ROLLUP(student,subject)

7-45

7-45

```

SELECT student, subject, AVG(score)::numeric(10,2)
FROM test_scores
WHERE student IN ('leo','regina')
GROUP BY ROLLUP (student,subject)
ORDER BY student NULLS LAST, subject NULLS LAST;

```

| student | subject | avg |
|---------|-----------|-------|
| leo | algebra | 82.00 |
| leo | calculus | 65.50 |
| leo | chemistry | 75.50 |
| leo | physics | 72.00 |
| leo | NULL | 73.75 |
| regina | algebra | 72.50 |
| regina | calculus | 64.50 |
| regina | chemistry | 73.50 |
| regina | economics | 90.00 |
| regina | physics | 84.00 |
| regina | NULL | 75.44 |
| NULL | NULL | 74.65 |

(12 rows)

Rollup은 그룹화하는 데 사용되는 SQL 집계 함수입니다. Rollup은 GROUP BY와 유사하지만, GROUP BY는 단일 그룹을 생성하는 반면, Rollup은 여러 그룹을 생성합니다. Rollup은 7-46을 참조하십시오.

7-46 Rollup을 사용하여 학생별 과목별 평균 점수를 계산하는 SQL 쿼리

```
SELECT student, subject, AVG(score)::numeric(10,2)
FROM test_scores
WHERE student IN ('leo','regina')
GROUP BY ROLLUP (subject,student)
ORDER BY student NULLS LAST, subject NULLS LAST;
```

| student | subject | avg |
|---------|-----------|-------|
| leo | algebra | 82.00 |
| leo | calculus | 65.50 |
| leo | chemistry | 75.50 |
| leo | physics | 72.00 |
| regina | algebra | 72.50 |
| regina | calculus | 64.50 |
| regina | chemistry | 73.50 |
| regina | economics | 90.00 |
| regina | physics | 84.00 |
| NULL | algebra | 77.25 |
| NULL | calculus | 65.00 |
| NULL | chemistry | 74.50 |
| NULL | economics | 90.00 |
| NULL | physics | 78.00 |
| NULL | NULL | 74.65 |

(15 rows)

Rollup은 GROUP BY와 유사하지만, Rollup은 여러 그룹을 생성합니다. Rollup은 7-46을 참조하십시오. Rollup은 GROUPING SETS((student), (student, subject), (subject), ())과 CUBE(student, subject)를 사용하여 사용할 수 있습니다. 7-47을 참조하십시오.

7-47 Rollup을 사용하여 학생별 과목별 평균 점수를 계산하는 SQL 쿼리

| student | subject | avg |
|---------|-----------|-------|
| leo | algebra | 82.00 |
| leo | calculus | 65.50 |
| leo | chemistry | 75.50 |
| leo | physics | 72.00 |
| leo | NULL | 73.75 |
| regina | algebra | 72.50 |
| regina | calculus | 64.50 |
| regina | chemistry | 73.50 |
| regina | economics | 90.00 |
| regina | physics | 84.00 |
| regina | NULL | 75.44 |
| NULL | algebra | 77.25 |
| NULL | calculus | 65.00 |
| NULL | chemistry | 74.50 |
| NULL | economics | 90.00 |
| NULL | physics | 78.00 |
| NULL | NULL | 74.65 |

PostgreSQL 数据库是 SQL 数据库。它支持 SQL 语言，并提供了许多高级功能，如事务、并发控制、安全性和性能优化等。PostgreSQL 数据库是一个开源的数据库系统，广泛应用于各种企业级应用中。

SQL procedural language
PL SQL PostgreSQL
SQL C PL/pgSQL PL/Perl
PL/Python PostgreSQL PL/V8
JavaScript PL/V8 Web JSON

JSONB 是 JavaScript 对象表示法 JSON 的 JSONB 二进制表示
5.6 节

除了 PL/R、PL/Java、PL/sh 和 PL/TSQL 之外，PostgreSQL 还支持
PL/Python、PL/Perl、PL/Scheme 和 PL/OpenCL。这些语言都支持
“用户自定义函数”的概念，PostgreSQL 将它们称为“用户自定义函数”。

8.1 PostgreSQL 用户自定义函数

PostgreSQL 用户自定义函数（UDF）允许用户创建自己的函数，
这些函数可以调用其他函数、过程和语言。

8.1.1 用户自定义函数

PostgreSQL 用户自定义函数（UDF）允许用户创建自己的函数，
这些函数可以调用其他函数、过程和语言。8-1
节

8-1 用户自定义函数

```
CREATE OR REPLACE FUNCTION func_name(arg1 arg1_datatype DEFAULT  
arg1_default)  
RETURNS some_type | set of some_type | TABLE (..) AS  
$$  
BODY of function  
$$  
LANGUAGE language_of_function
```

用户自定义函数（UDF）允许用户创建自己的函数，
这些函数可以调用其他函数、过程和语言。

用户自定义函数（UDF）允许用户创建自己的函数，
这些函数可以调用其他函数、过程和语言。
用户自定义函数（UDF）允许用户创建自己的函数，
这些函数可以调用其他函数、过程和语言。

```
big_elephant(ear_size numeric, skin_color text DEFAULT 'blue',  
name text DEFAULT 'Dumbo')
```

ear_size skin_color
\$1 \$2 \$3

```
big_elephant(name => 'Wooly', ear_size => 1.2)
```

big_elephant(1.2, 'blue', 'Wooly')
big_elephant
skin_color 'blue' name
name
skin_color



PostgreSQL 9.5 name => 'Wooly'
PostgreSQL 9.4 name := 'Wooly'
arg1_name := arg1_value
PostgreSQL 9.5

LANGUAGE

database
SELECT lanname FROM pg_language;

VOLATILITY

IMMUTABLE

データベースのテーブルは、**IMMUTABLE** の
テーブルは、**IMMUTABLE** の
テーブル

STABLE のテーブル

データベースのテーブルは、**SQL** の
データベースのテーブル

VOLATILE のテーブル

データベースのテーブルは、**VOLATILE** の
データベースのテーブル

データベースの**VOLATILITY** の
データベースの**VOLATILE** の
データベースの
データベース

STRICT のテーブル

データベースの**NULL** の
NULL の**STRICT** の
STRICT の“**STRICT on SQL Functions**”の

COST のテーブル

データベースの**SQL** の **PL/pgSQL** の
データベース **100** の **C** の **1** の **WHERE** の
データベースの

ROWS のテーブル

データベースの
データベースの

SECURITY DEFINER □□□□□□ □

[illegible]

PARALLEL □□□□ □

PostgreSQL 9.6

PARALLEL UNSAFE

"

SAFE

不可变 IMMUTABLE
安全 SAFE

UNSAFE

[illegible]

RESTRICTED

[illegible]

PARALLEL PostgreSQL 9.6
PARALLEL

8.1.2 □□□□□□□□

PostgreSQL

```

SQL
UPDATE 1500
1500

```































BEFORE AFTER INSTEAD OF
 BEFORE
 AFTER
 INSTEAD OF
 BEFORE AFTER
 INSTEAD OF

BEFORE

AFTER

When you execute the following command, PostgreSQL will return the error message "UPDATE OF + column". This is because the column name is not specified in the UPDATE statement.

```
7-5
```

PostgreSQL                              

```

##### trigger #####PostgreSQL #####
#####
#####
#####

```

PostgreSQL

本書は PostgreSQL のインストール、アップグレード、バックアップ、リカバリ、SQL の最適化、
 拡張機能 SQL 関数、PL/pgSQL、PostgreSQL のインストール、アップグレード、バックアップ、
 リカバリ、8.3.2 のインストール、PL/pgSQL のインストール、アップグレード、バックアップ、リカバリ、

8.1.3 □□□□

ANSI SQL MIN MAX AVG SUM COUNT PostgreSQL PostgreSQL 7.3

PostgreSQL SQL
Aggregates” PL/pgSQL PL/Python SQL

[illegible]

```
CREATE AGGREGATE my_agg (input data type) (  
SFUNC=state function name,  
STYPE=state type,  
FINALFUNC=final function name,  
INITCOND=initial state value, SORTOP=sort_operator  
);
```

[illegible]

```

000 SORTOP 000000000000 > 0 < 0000000000000000 MAX 0MIN
00000000000000000000 SORTOP 0000000000000000 MAX 0MIN 0
000000000000000000000000000000000000000000000000000 MAX 0MIN 00000
000000000000000000000000000000000000000000000000000 SORTOP 0000000000
000000000000000000000000000000000000000000000000000

```

□□□□□□□ SQL □□□□□□□□□□□□□□□□□□□□ 8-3 □□□

□□ **8-3** □□□□□□□□□□ SQL □□

```
CREATE OR REPLACE FUNCTION
update_logs(log_id int, param_user_name varchar, param_description
text)
RETURNS void AS
$$
UPDATE logs SET user_name = $2, description = $3
, log_ts = CURRENT_TIMESTAMP WHERE log_id = $1;
$$
LANGUAGE 'sql' VOLATILE;
```

□□□□□□□□□□□□

```
SELECT update_logs(12, 'alex', 'Fell back asleep.');
```

□□□□□□□□□□□□□□□□□□□□ SQL □□□□□□□□□□□□□□□□□□□□
ANSI SQL □□□□□ RETURNS TABLE □□□□□□□□ OUT □□□□□□□□□□
□□□□□□□□□□□□□□□□ RETURNS TABLE □□□□□□□□□□□□ 8-4 □
□□□□□□□□□□□□□□□□□□□□

□□ **8-4** □□□□□□□□

□□ RETURNS TABLE □□□□□□□□□□

```
CREATE OR REPLACE FUNCTION select_logs_rt(param_user_name varchar)
RETURNS TABLE (log_id int, user_name varchar(50),
description text, log_ts timestampz) AS
$$
SELECT log_id, user_name, description, log_ts FROM logs WHERE
user_name = $1;
$$
LANGUAGE 'sql' STABLE PARALLEL SAFE;
```

□□ OUT □□□□□□□□□□

```
CREATE OR REPLACE FUNCTION select_logs_out(param_user_name varchar,
OUT log_id int
, OUT user_name varchar, OUT description text, OUT log_ts
timestampz)
RETURNS SETOF record AS
$$
SELECT * FROM logs WHERE user_name = $1;
$$
LANGUAGE 'sql' STABLE PARALLEL SAFE;
```

□□□□□□□□□□□□□□□□

```
CREATE OR REPLACE FUNCTION select_logs_so(param_user_name varchar)
RETURNS SETOF logs AS
$$
SELECT * FROM logs WHERE user_name = $1;
$$
LANGUAGE 'sql' STABLE PARALLEL SAFE;
```

□□□□□□□□□□□□□□□□□□□□

```
SELECT * FROM select_logs_xxx('alex');
```

8.2.2 □□SQL□□□□□□□□

□□□□□□□□PostgreSQL □□□□□□□□ MIN □MAX □COUNT □AVG □□□□□
□□□□□□□□□□□□□□□□□□ SQL □□□□□□□□□□□□□□□□□□□□□□□□□□
□□ n □□□□□□□□ n □□□□ $(x_1 * x_2 * x_3 \dots x_n)^{(1/n)}$ □□□□□□□□□□□□
□□
□□
□□
□□

□□ 8-5
5□□ 0□

□□ 8-5 □□□□□□□□□□□□□□□□□□□□

```
CREATE OR REPLACE FUNCTION geom_mean_state(prev numeric[2], next
numeric)
RETURNS numeric[2] AS
$$
SELECT
CASE
WHEN $2 IS NULL OR $2 = 0 THEN $1
ELSE ARRAY[COALESCE($1[1],0) + ln($2), $1[2] + 1]
END;
$$
LANGUAGE sql IMMUTABLE PARALLEL SAFE;
```

1. 如果 \$2 是 NULL 或 0，那么返回 \$1。
 2. 否则，返回一个数组，第一个元素是 \$1[1] 与 \$2 的自然对数的和，第二个元素是 \$1[2] 加 1。

8-6 计算几何平均数的函数

8-6 计算几何平均数的函数

```
CREATE OR REPLACE FUNCTION geom_mean_final(numeric[2])
RETURNS numeric AS
$$
SELECT CASE WHEN $1[2] > 0 THEN exp($1[1]/$1[2]) ELSE 0 END;
$$
LANGUAGE sql IMMUTABLE PARALLEL SAFE;
```

8-7 计算几何平均数的函数

8-7 计算几何平均数的函数

```
CREATE AGGREGATE geom_mean(numeric) (
SFUNC=geom_mean_state,
STYPE=numeric[],
FINALFUNC=geom_mean_final,
PARALLEL = safe,
INITCOND='{0,0}'
);
```

8-8 5

8-8 5

```
SELECT left(tract_id,5) As county, geom_mean(val) As div_county
FROM census.vw_facts
WHERE category = 'Population' AND short_name != 'white_alone'
GROUP BY county
ORDER BY div_county DESC LIMIT 5;
```

| county | div_county |
|--------|---------------------|
| 25025 | 85.1549046212833364 |
| 25013 | 79.5972921427888918 |
| 25017 | 74.7697097102419689 |
| 25021 | 73.8824162064128504 |
| 25027 | 73.5955049035237656 |

8-9

8-9 5

```
WITH X AS (SELECT
  tract_id,
  left(tract_id,5) As county,
  geom_mean(val) OVER (PARTITION BY tract_id) As div_tract,
  ROW_NUMBER() OVER (PARTITION BY tract_id) As rn,
  geom_mean(val) OVER(PARTITION BY left(tract_id,5)) As div_county
FROM census.vw_facts WHERE category = 'Population' AND short_name
!= 'white_alone'
)
SELECT tract_id, county, div_tract, div_county
FROM X
WHERE rn = 1
ORDER BY div_tract DESC, div_county DESC LIMIT 5;
```

| tract_id | county | div_tract | div_county |
|-------------|--------|----------------------|---------------------|
| 25025160101 | 25025 | 302.6815688785928786 | 85.1549046212833364 |
| 25027731900 | 25027 | 265.6136902148147729 | 73.5955049035237656 |
| 25021416200 | 25021 | 261.9351057509603296 | 73.8824162064128504 |
| 25025130406 | 25025 | 260.3241378371627137 | 85.1549046212833364 |

UPDATE 与 DELETE 语句的触发器。PL/PythonU 与 PL/PerlU 语句的触发器。PL/pgSQL 语句的触发器。

图 8-11 数据库触发器

图 8-11 数据库触发器

```
CREATE OR REPLACE FUNCTION trig_time_stamper() RETURNS trigger AS
❶
$$
BEGIN
    NEW.upd_ts := CURRENT_TIMESTAMP;
    RETURN NEW;
END;
$$
LANGUAGE plpgsql VOLATILE;

CREATE TRIGGER trig_1
BEFORE INSERT OR UPDATE OF session_state, session_id ❷
ON web_sessions
FOR EACH ROW EXECUTE PROCEDURE trig_time_stamper();
```

❶ 该函数将 NEW.upd_ts 的值设置为 CURRENT_TIMESTAMP 的值。

❷ “BEFORE” 9.0 版本之前，BEFORE 触发器只能在 UPDATE 或 INSERT 语句执行之前执行。在 9.0 版本中，BEFORE 触发器也可以在 UPDATE 或 INSERT 语句执行之前执行。在 9.0 版本中，BEFORE 触发器也可以在 UPDATE 或 INSERT 语句执行之前执行。在 9.0 版本中，BEFORE 触发器也可以在 UPDATE 或 INSERT 语句执行之前执行。

8.4 PL/Python 触发器

Python 触发器是 PostgreSQL 数据库中的一种触发器。Python 触发器是 PostgreSQL 数据库中的一种触发器。Python 2 与 Python 3 兼容。



データベース database 拡張 PL/Python2U
 PL/Python3U 拡張 PL/Python2U 拡張 PL/Python3U 拡張
 PL/PythonU 拡張 PL/Python2U 拡張

PL/Python Python Windows
 Mac Python <http://www.python.org/download/>
 Linux/Unix Python PostgreSQL PL/Python Python PostgreSQL Python

```
CREATE EXTENSION plpython2u;  
CREATE EXTENSION plpython3u;
```

PostgreSQL Python Python

PostgreSQL PL/PythonU Python
 Python plpythonu
 plpython2u Python 2.7
 Python 2.7

Python

PostgreSQL PostgreSQL Python
 PL/Python PL/Python
 PostgresOnline PL/Python

Python PL/pgSQL 8-12
 PL/Python PostgreSQL

8-12 PL/Python PostgreSQL


```

CREATE OR REPLACE FUNCTION postgresql_help_search(param_search
text)
RETURNS text AS
$$
import urllib, re ❶
response = urllib.urlopen(
    'http://www.postgresql.org/search/?u=%2Fdocs%2Fcurrent%2F&q=' +
param_search
) ❷
raw_html = response.read() ❸
result =
    raw_html[raw_html.find("<!-- docbot goes here -->") :
    raw_html.find("<!-- pgContentWrap -->") - 1] ❹
result = re.sub('<[^\>]+?>', '', result).strip() ❺
return result ❻
$$
LANGUAGE plpython2u SECURITY DEFINER STABLE;

```

❶ ❶ import urllib, re

❷ ❷ response = urllib.urlopen(

❸ ❸ raw_html = response.read()

❹ ❹ raw_html[raw_html.find("<!-- docbot goes here -->") :
pgContentWrap -->] - 1]

❺ ❺ result = re.sub('<[^\>]+?>', '', result).strip()

❻ ❻ return result

Python ❷-13 ❷-12
❷-12

❷-13 Python

```

SELECT search_term, left(postgresql_help_search(search_term),125)
As result
FROM (VALUES ('regex_match'),('pg_trgm'),('tsvector')) As
x(search_term);

```

PL/Python 8-14 PL/Python PostgreSQL postgres postgres

8-14

```
CREATE OR REPLACE FUNCTION list_incoming_files()  
RETURNS SETOF text AS  
$$  
import os  
return os.listdir('/incoming')  
$$  
LANGUAGE 'plpython2u' VOLATILE SECURITY DEFINER;
```

```
SELECT filename  
FROM list_incoming_files() As filename  
WHERE filename ILIKE '%.csv'
```

8.5 PL/V8 PL/CoffeeScript PL/LiveScript

PL/V8 PL/JavaScript Google V8 JavaScript JSON PL/V8 PostgreSQL PostgreSQL Windows PostgreSQL 9.6 PL/V8 32 64

PostgreSQL PL/V8 JavaScript

PL/V8 plv8

JavaScript 入門

PL/CoffeeScript `plcoffee`

CoffeeScript JavaScript JavaScript Python Python

PL/LiveScript plls

LiveScript vs CoffeeScript

“CoffeeScript: 10 Reasons to Switch from CoffeeScript to LiveScript”

LiveScript vs CoffeeScript

CoffeeScript vs LiveScript

Python vs F#

Haskell

PL/Python

LiveScript

PL/CoffeeScript PL/LiveScript PL/V8

8-15 database

□□ **8-15** PL/V8 □□□□□□□□

```
CREATE EXTENSION plv8;
CREATE EXTENSION plcoffee;
CREATE EXTENSION plls;
```

☐ PL/pgSQL
 ☐ PL/V8
 ☐ PL/R
 ☐ PL/Python
 ☐ PL/Perl
 ☐ PL/Java
 ☐ PL/JavaScript
 ☐ PL/Python2
 ☐ PL/Python3
 ☐ PL/Perl5
 ☐ PL/Python4
 ☐ PL/Python5
 ☐ PL/Python6
 ☐ PL/Python7
 ☐ PL/Python8
 ☐ PL/Python9
 ☐ PL/Python10
 ☐ PL/Python11
 ☐ PL/Python12
 ☐ PL/Python13
 ☐ PL/Python14
 ☐ PL/Python15
 ☐ PL/Python16
 ☐ PL/Python17
 ☐ PL/Python18
 ☐ PL/Python19
 ☐ PL/Python20
 ☐ PL/Python21
 ☐ PL/Python22
 ☐ PL/Python23
 ☐ PL/Python24
 ☐ PL/Python25
 ☐ PL/Python26
 ☐ PL/Python27
 ☐ PL/Python28
 ☐ PL/Python29
 ☐ PL/Python30
 ☐ PL/Python31
 ☐ PL/Python32
 ☐ PL/Python33
 ☐ PL/Python34
 ☐ PL/Python35
 ☐ PL/Python36
 ☐ PL/Python37
 ☐ PL/Python38
 ☐ PL/Python39
 ☐ PL/Python40
 ☐ PL/Python41
 ☐ PL/Python42
 ☐ PL/Python43
 ☐ PL/Python44
 ☐ PL/Python45
 ☐ PL/Python46
 ☐ PL/Python47
 ☐ PL/Python48
 ☐ PL/Python49
 ☐ PL/Python50
 ☐ PL/Python51
 ☐ PL/Python52
 ☐ PL/Python53
 ☐ PL/Python54
 ☐ PL/Python55
 ☐ PL/Python56
 ☐ PL/Python57
 ☐ PL/Python58
 ☐ PL/Python59
 ☐ PL/Python60
 ☐ PL/Python61
 ☐ PL/Python62
 ☐ PL/Python63
 ☐ PL/Python64
 ☐ PL/Python65
 ☐ PL/Python66
 ☐ PL/Python67
 ☐ PL/Python68
 ☐ PL/Python69
 ☐ PL/Python70
 ☐ PL/Python71
 ☐ PL/Python72
 ☐ PL/Python73
 ☐ PL/Python74
 ☐ PL/Python75
 ☐ PL/Python76
 ☐ PL/Python77
 ☐ PL/Python78
 ☐ PL/Python79
 ☐ PL/Python80
 ☐ PL/Python81
 ☐ PL/Python82
 ☐ PL/Python83
 ☐ PL/Python84
 ☐ PL/Python85
 ☐ PL/Python86
 ☐ PL/Python87
 ☐ PL/Python88
 ☐ PL/Python89
 ☐ PL/Python90
 ☐ PL/Python91
 ☐ PL/Python92
 ☐ PL/Python93
 ☐ PL/Python94
 ☐ PL/Python95
 ☐ PL/Python96
 ☐ PL/Python97
 ☐ PL/Python98
 ☐ PL/Python99
 ☐ PL/Python100
 ☐ PL/Python101
 ☐ PL/Python102
 ☐ PL/Python103
 ☐ PL/Python104
 ☐ PL/Python105
 ☐ PL/Python106
 ☐ PL/Python107
 ☐ PL/Python108
 ☐ PL/Python109
 ☐ PL/Python110
 ☐ PL/Python111
 ☐ PL/Python112
 ☐ PL/Python113
 ☐ PL/Python114
 ☐ PL/Python115
 ☐ PL/Python116
 ☐ PL/Python117
 ☐ PL/Python118
 ☐ PL/Python119
 ☐ PL/Python120
 ☐ PL/Python121
 ☐ PL/Python122
 ☐ PL/Python123
 ☐ PL/Python124
 ☐ PL/Python125
 ☐ PL/Python126
 ☐ PL/Python127
 ☐ PL/Python128
 ☐ PL/Python129
 ☐ PL/Python130
 ☐ PL/Python131
 ☐ PL/Python132
 ☐ PL/Python133
 ☐ PL/Python134
 ☐ PL/Python135
 ☐ PL/Python136
 ☐ PL/Python137
 ☐ PL/Python138
 ☐ PL/Python139
 ☐ PL/Python140
 ☐ PL/Python141
 ☐ PL/Python142
 ☐ PL/Python143
 ☐ PL/Python144
 ☐ PL/Python145
 ☐ PL/Python146
 ☐ PL/Python147
 ☐ PL/Python148
 ☐ PL/Python149
 ☐ PL/Python150
 ☐ PL/Python151
 ☐ PL/Python152
 ☐ PL/Python153
 ☐ PL/Python154
 ☐ PL/Python155
 ☐ PL/Python156
 ☐ PL/Python157
 ☐ PL/Python158
 ☐ PL/Python159
 ☐ PL/Python160
 ☐ PL/Python161
 ☐ PL/Python162
 ☐ PL/Python163
 ☐ PL/Python164
 ☐ PL/Python165
 ☐ PL/Python166
 ☐ PL/Python167
 ☐ PL/Python168
 ☐ PL/Python169
 ☐ PL/Python170
 ☐ PL/Python171
 ☐ PL/Python172
 ☐ PL/Python173
 ☐ PL/Python174
 ☐ PL/Python175
 ☐ PL/Python176
 ☐ PL/Python177
 ☐ PL/Python178
 ☐ PL/Python179
 ☐ PL/Python180
 ☐ PL/Python181
 ☐ PL/Python182
 ☐ PL/Python183
 ☐ PL/Python184
 ☐ PL/Python185
 ☐ PL/Python186
 ☐ PL/Python187
 ☐ PL/Python188
 ☐ PL/Python189
 ☐ PL/Python190
 ☐ PL/Python191
 ☐ PL/Python192
 ☐ PL/Python193
 ☐ PL/Python194
 ☐ PL/Python195
 ☐ PL/Python196
 ☐ PL/Python197
 ☐ PL/Python198
 ☐ PL/Python199
 ☐ PL/Python200
 ☐ PL/Python201
 ☐ PL/Python202
 ☐ PL/Python203
 ☐ PL/Python204
 ☐ PL/Python205
 ☐ PL/Python206
 ☐ PL/Python207
 ☐ PL/Python208
 ☐ PL/Python209
 ☐ PL/Python210
 ☐ PL/Python211
 ☐ PL/Python212
 ☐ PL/Python213
 ☐ PL/Python214
 ☐ PL/Python215
 ☐ PL/Python216
 ☐ PL/Python217
 ☐ PL/Python218
 ☐ PL/Python219
 ☐ PL/Python220
 ☐ PL/Python221
 ☐ PL/Python222
 ☐ PL/Python223
 ☐ PL/Python224
 ☐ PL/Python225
 ☐ PL/Python226
 ☐ PL/Python227
 ☐ PL/Python228
 ☐ PL/Python229
 ☐ PL/Python230
 ☐ PL/Python231
 ☐ PL/Python232
 ☐ PL/Python233
 ☐ PL/Python234
 ☐ PL/Python235
 ☐ PL/Python236
 ☐ PL/Python237
 ☐ PL/Python238
 ☐ PL/Python239
 ☐ PL/Python240
 ☐ PL/Python241
 ☐ PL/Python242
 ☐ PL/Python243
 ☐ PL/Python244
 ☐ PL/Python245
 ☐ PL/Python246
 ☐ PL/Python247
 ☐ PL/Python248
 ☐ PL/Python249
 ☐ PL/Python250
 ☐ PL/Python251
 ☐ PL/Python252
 ☐ PL/Python253
 ☐ PL/Python254
 ☐ PL/Python255
 ☐ PL/Python256
 ☐ PL/Python257

- SQL vs PL/pgSQL
- SQL vs PL/pgSQL vs PL/Python vs PL/R vs C

- 数据库函数
- 数据库函数使用 try-catch 语句
- 使用 eval 调用 JavaScript 函数
- 使用 JSON 函数处理 JSON 数据
- 使用 DO 语句执行 SQL 语句
- 使用 Node.js/PL/V8 使用 Node.js 编写的 V8 函数
Node.js 函数使用 PL/V8 使用 Node.js 编写的 JavaScript 函数
PostgreSQL 使用 plv8x 使用 Node.js 使用 PL/V8 函数

PostgresOnline 数据库函数 PL/V8 函数
使用 JavaScript 函数 PL/V8 函数“Using PLV8 to Build JSON Selectors”
使用 PL/V8 函数 Web 函数
使用 JavaScript 函数 PostgreSQL 函数
使用 PostgreSQL 函数

8.5.1 数据库函数

PL/V8 函数使用 PL/V8 函数 JavaScript 函数
使用 JavaScript 函数
使用 PL/V8 函数 8-16 图

图 8-16 使用 PL/V8 函数

```
CREATE OR REPLACE FUNCTION
validate_email(email text) returns boolean as
$$
  var re = /\S+@\S+\.\S+/;
  return re.test(email);
$$ LANGUAGE plv8 IMMUTABLE STRICT PARALLEL SAFE;
```

使用 JavaScript 函数 8-17 图

图 8-17 使用 PL/V8 函数

```
SELECT email, validate_email(email) AS is_valid
FROM (VALUES ('alexgomezq@gmail.com'))
```

```
,('alexgomezqgmail.com'),('alexgomezq@gmailcom')) AS x (email);
```

| | | | | | | |
|--|--|--|--|--|--|--|
| | | | | | | |
|--|--|--|--|--|--|--|

| email | is_valid |
|----------------------|----------|
| alexgomezq@gmail.com | t |
| alexgomezqgmail.com | f |
| alexgomezq@gmailcom | f |

PL/pgSQL PostgreSQL PL/V8 Web

0000000000 text 000000 text 000000 JavaScript 0000000000
 000000000000000000000000 PostgreSQL 000000000000000000
 0000000000 PL/V8 000000000000000000000000 Andrew
 Dunstan 000“Loading Useful Modules in PLV8”00000
 JavaScript 000000000000PL/V8 00000 eval 0000000000000000
 0 JavaScript 0000000000000000000000000000

js2coffee.org 8-17 JavaScript
CoffeeScript 8-18

8-18 PL/Coffee

```
CREATE OR REPLACE FUNCTION
validate_email(email text) returns boolean as
$$
    re = /\S+@\S+\.\S+/
    return re.test email
$$
LANGUAGE plcoffee IMMUTABLE STRICT PARALLEL SAFE;
```

CoffeeScript → JavaScript → PL/V8
→ LiveScript → CoffeeScript → PL/V8
LANGUAGE plls →

8.5.2 PL/V8

図 8-19 図 8-20 例 PL/V8 関数
図 8.2.2 例

図 8-19 PL/V8 関数

```
CREATE OR REPLACE FUNCTION geom_mean_state(prev numeric[2], next
numeric)
RETURNS numeric[2] AS
$$
    return (next == null || next== 0) ? prev :
    [(prev[0] == null)? 0: prev[0] + Math.log(next), prev[1] + 1];
$$
LANGUAGE plv8 IMMUTABLE PARALLEL SAFE;
```

図 8-20 PL/V8 関数

```
CREATE OR REPLACE FUNCTION geom_mean_final(in_num numeric[2])
RETURNS numeric AS
$$
    return in_num[1] > 0 ? Math.exp(in_num[0]/in_num[1]) : 0;
$$
LANGUAGE plv8 IMMUTABLE PARALLEL SAFE;
```

図 CREATE AGGREGATE 関数
図 8-21 例

図 8-21 PL/V8 関数

```
CREATE AGGREGATE geom_mean(numeric) (
    SFUNC=geom_mean_state,
    STYPE=numeric[],
    FINALFUNC=geom_mean_final,
    PARALLEL = safe,
    INITCOND='{0,0}'
```

| | |
|---|--|
|) | |
|---|--|

8-9 geom_mean PL/V8
SQL geom_mean PL/V8
SQL PL/V8
SQL 10 20

8.5.3 PL/V8

PostgreSQL 7.3 数据库系统安装与使用

PostgreSQL PL/pgSQL SQL PL/Python PL/Perl C PL/R PL/V8 C C

```
PL/V8 plv8.window_object PL/V8
PL/V8
```

```
0000 8-22 00000000000000000000000000000000"0000"00
0000000000 true0000 false0000"0000"000000000000 N
000000000000000000000000000000000000"0000"ofs 00
00000000
```

□□ **8-22** □ PL/V8 □□□□□□□□□□□□□□□□

```
CREATE FUNCTION run_begin(arg anyelement, ofs int) RETURNS boolean  
AS $$  
    var winobj = plv8.get_window_object();  
    var result = true;  
    /**      */  
    var cval = winobj.get_func_arg_in_partition(0,  
                                                0,  
                                                winobj.SEEK_CURRENT,  
                                                false);  
  
    for (i = 1; i < ofs; i++){
```

```

    /**  */
    nval = winobj.get_func_arg_in_partition(0,
                                           i,
winobj.SEEK_CURRENT,
                                           false);

    result = (cval == nval) ? true : false;
    if (!result){
        break;
    }
    /**  */
    cval = nval;

}
return result;
$$ LANGUAGE plv8 WINDOW;

```

PL/V8 8-22

PL/V8 API PL/V8 “PL/V8 API” ofs ofs true false PL/V8 get_func_arg_in_partiton false

8-23

8-23 PL/V8

```

SELECT id, player, toss,
       run_begin(toss,3) OVER (PARTITION BY player ORDER BY id) AS rb
FROM coin_tosses
ORDER BY player, id;

```

| id | player | toss | rb |
|----|--------|------|----|
| 4 | alex | H | t |
| 8 | alex | H | t |
| 12 | alex | H | f |

| | | | |
|----|--------|---|---|
| 16 | alex | H | f |
| 2 | leo | T | f |
| 6 | leo | H | f |
| 10 | leo | H | f |
| 14 | leo | T | f |
| 1 | regina | H | f |
| 5 | regina | H | f |
| 9 | regina | T | f |
| 13 | regina | T | f |
| 3 | sonia | T | t |
| 7 | sonia | T | t |
| 11 | sonia | T | f |
| 15 | sonia | T | f |

(16 rows)

本书附带的 PL/V8 脚本可在 GitHub 上“window regression script”脚本库中找到。PL/V8 脚本 PostgreSQL 脚本 lead lag row_number cume_dist first_value last_value 脚本

第 9 章 窗口函数

本章将介绍窗口函数。窗口函数是 SQL 中一类特殊的函数，它可以在 PostgreSQL 中用于对数据进行窗口分析。窗口函数是 SQL 中一类特殊的函数，它可以在 SQL 中用于对数据进行窗口分析。窗口函数是 SQL 中一类特殊的函数，它可以在 SQL 中用于对数据进行窗口分析。

9.1 使用 EXPLAIN 分析查询

本章将介绍如何使用 EXPLAIN 和 EXPLAIN (ANALYZE) 来分析查询。PostgreSQL 提供了 EXPLAIN 和 EXPLAIN (ANALYZE) 来分析查询。EXPLAIN 和 EXPLAIN (ANALYZE) 可以返回查询计划的 XML、JSON 或 YAML 格式。

pgAdmin の EXPLAIN 機能について説明します。

9.1.1 EXPLAIN

EXPLAIN SQL 文を実行すると、EXPLAIN 結果が表示されます。

- EXPLAIN SQL 文を実行すると、SQL 文の実行計画が表示されます。
- ANALYZE EXPLAIN (ANALYZE) は、SQL 文の実行計画と実行時間を表示します。
- EXPLAIN VERBOSE EXPLAIN (VERBOSE) は、EXPLAIN 結果の詳細な説明を表示します。
- ANALYZE BUFFERS EXPLAIN (ANALYZE, BUFFERS) は、EXPLAIN 結果とバッファの使用状況を表示します。

SQL 文を実行する前に EXPLAIN (ANALYZE, VERBOSE, BUFFERS) + を実行すると、実行計画と実行時間、およびバッファの使用状況が表示されます。

UPDATE 文、INSERT 文、DML 文を実行する前に EXPLAIN (ANALYZE) を実行すると、実行計画と実行時間が表示されます。BEGIN 文、ROLLBACK 文は除外されます。

pgAdmin の EXPLAIN 機能は、pgAdmin の EXPLAIN 機能と EXPLAIN (ANALYZE) 機能とを統合しています。

9.1.2 実行計画

EXPLAIN (ANALYZE) SQL 文を実行すると、4-1 のように 4-2 のように表示されます。

実行計画は、SQL 文の実行順序と実行時間、およびバッファの使用状況を示します。

```
ALTER TABLE census.hisp_pop DROP CONSTRAINT IF EXISTS  
hisp_pop_pkey;
```

9-1

9-1 EXPLAIN (ANALYZE)

```
EXPLAIN (ANALYZE)
SELECT tract_id, hispanic_or_latino
FROM census.hisp_pop
WHERE tract_id = '25025010103';
```

```

000000 EXPLAIN 00000000000000000000000000000000 ANALYZE 000000
0 EXPLAIN (ANALYZE) 00000000000000000000000000000000

```

□□ 9-2 □□□ 9-1 □□□□□□□□

□□ **9-2** EXPLAIN (ANALYZE) □□□□□

```
Seq Scan on hisp_pop
(cost=0.00..33.48 rows=1 width=16)
(actual time=0.213..0.346 rows=1 loops=1)
Filter: ((tract_id)::text = '25025010103'::text)
Rows Removed by Filter: 1477
Planning time: 0.095 ms
Execution time: 0.381 ms
```

EXPLAIN

cost=0.00..33.48 rows=9-2

33.84

0

[illegible]

Figure 9-1 Analyze query results

Figure 9-2 Rows Removed by Filter:1477

PostgreSQL 9.4 EXPLAIN

```
ALTER TABLE census.hisp_pop ADD CONSTRAINT hisp_pop_pkey PRIMARY
KEY(tract_id);
```

Figure 9-1 Figure 9-3

9-3 EXPLAIN (ANALYZE)

```
Index Scan using idx_hisp_pop_tract_id_pat on hisp_pop
(cost=0.28..8.29 rows=1 width=16)
(actual time=0.018..0.019 rows=1 loops=1)
Index Cond: ((tract_id)::text = '25025010103'::text)
Planning time: 0.110 ms
Execution time: 0.046 ms
```

33.48 8.29 0
1477

Figure 9-4

9-4 GROUP BY SUM EXPLAIN (ANALYZE)

```
EXPLAIN (ANALYZE)
SELECT left(tract_id,5) AS county_code, SUM(white_alone) As w
FROM census.hisp_pop
WHERE tract_id BETWEEN '25025000000' AND '25025999999'
GROUP BY county_code;
```

9-5 9-4

9-5 EXPLAIN (ANALYZE)

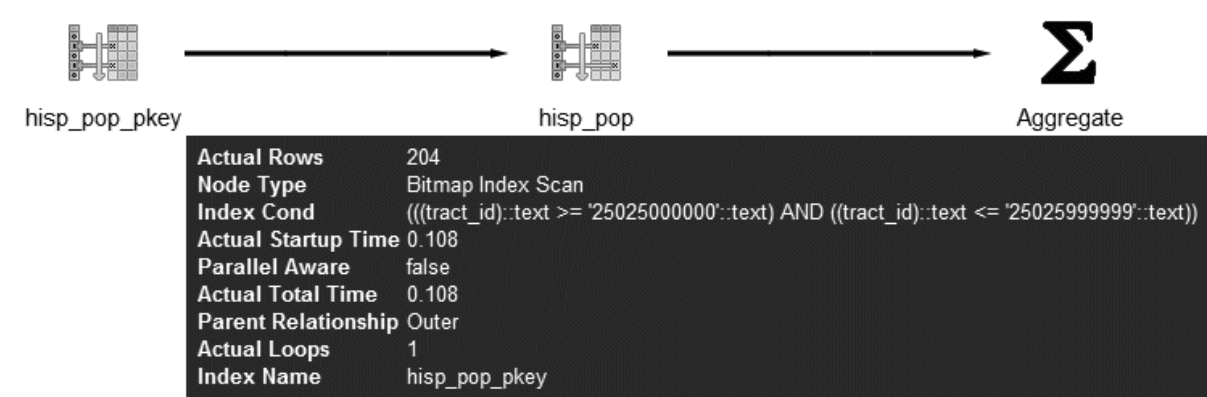
```
HashAggregate
  (cost=29.57..32.45 rows=192 width=16)
  (actual time=0.664..0.664 rows=1 loops=1)
  Group Key: "left"((tract_id)::text, 5)
  -> Bitmap Heap Scan on hisp_pop
    (cost=10.25..28.61 rows=192 width=16)
    (actual time=0.441..0.550 rows=204 loops=1)
    Recheck Cond:
      (((tract_id)::text >= '25025000000'::text) AND
       ((tract_id)::text <= '25025999999'::text))
    Heap Blocks: exact=15
    -> Bitmap Index Scan on hisp_pop_pkey
      (cost=0.00..10.20 rows=192 width=0)
      (actual time=0.421..0.421 rows=204 loops=1)
      Index Cond:
        (((tract_id)::text >= '25025000000'::text) AND
         ((tract_id)::text <= '25025999999'::text))
Planning time: 4.835 ms
Execution time: 0.732 ms
```

9-5 PostgreSQL

```
Planning time: 0.200 ms
Execution time: 0.635 ms
```

9.1.3 索引扫描

9-1 EXPLAIN (ANALYZE) 索引扫描



9-1 索引扫描

<http://explain.depesz.com/> Hubert Lubaczewski
9-2

HTMLTEXTSTATS

Did it help? Consider supporting us

| node type | count | sum of times | % of query |
|-------------------|-------|--------------|------------|
| Bitmap Heap Scan | 1 | 0.129 ms | 19.4 % |
| Bitmap Index Scan | 1 | 0.421 ms | 63.4 % |
| HashAggregate | 1 | 0.114 ms | 17.2 % |

| Table name | Scan count | Total time | % of query |
|------------------|------------|--------------|------------|
| scan type | count | sum of times | % of table |
| hisp_pop | 1 | 0.129 ms | 19.4 % |
| Bitmap Heap Scan | 1 | 0.129 ms | 100.0 % |

PostgreSQL 9.5.4 pg_stat_statements 1.5.0

(1) postgresql.conf shared_preload_libraries =
"pg_stat_statements"

(2) postgresql.conf

```
pg_stat_statements.max = 10000  
pg_stat_statements.track = all
```

(3) postgresql

(4) SQL database CREATE
EXTENSION pg_stat_statements;

- pg_stat_statements database SQL
- pg_stat_statements_reset

9-6 postgresql_book database 5 SQL

9-6 database

```
SELECT  
    query, calls, total_time, rows,  
  
    100.0*shared_blks_hit/NULLIF(shared_blks_hit+shared_blks_read,0) AS  
    hit_percent  
FROM pg_stat_statements As s INNER JOIN pg_database As d On d.oid =  
s.dbid  
WHERE d.datname = 'postgresql_book'  
ORDER BY total_time DESC LIMIT 5;
```


9.3 数据库SQL

データベースの種類は、SQL データベースと NoSQL データベースに分類される。SQL データベースは、PostgreSQL などがある。

SQL 数据库系统 SQL 数据库系统
SQL 数据库系统 SQL 数据库系统
SQL 数据库系统 SQL 数据库系统
SQL 数据库系统

SQL PostgreSQL

SQL 데이터베이스는 SQL 쿼리를 사용하여 데이터를 저장하고 관리하는 데 사용됩니다. PostgreSQL은 오픈 소스 RDBMS입니다.

9.3.1 SELECT 语句

SQL 数据库系统，其核心是 SQL 语言。SQL 语言是一种非过程化的、面向集合的、自描述的语言。它由数据定义语言（DDL）、数据操纵语言（DML）和数据控制语言（DCL）组成。DDL 用于定义数据库结构，如表、视图、索引等；DML 用于操作数据库中的数据，如插入、更新、删除等；DCL 用于控制数据库访问权限。

9-7

9-7 **9-7**

```
SELECT tract_id,
       (SELECT COUNT(*) FROM census.facts As F
        WHERE F.tract_id = T.tract_id) As num_facts,
       (SELECT COUNT(*)
        FROM census.lu_fact_types As Y
        WHERE Y.fact_type_id IN (
            SELECT fact_type_id
            FROM census.facts F
            WHERE F.tract_id = T.tract_id
        )
)
```

```

) As num_fact_types
FROM census.lu_tracts As T;

```

SQL 9-8 SELECT

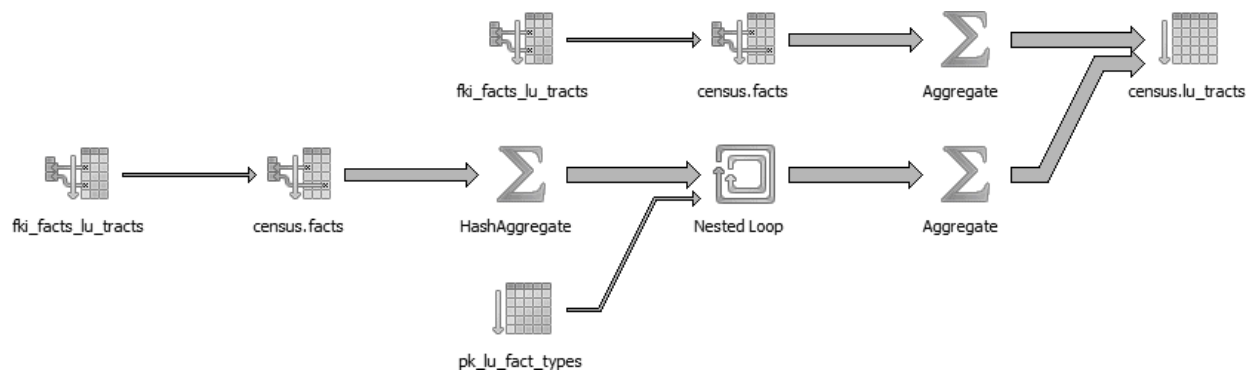
9-8

```

SELECT T.tract_id,
       COUNT(f.fact_type_id) As num_facts,
       COUNT(DISTINCT fact_type_id) As num_fact_types
FROM census.lu_tracts As T LEFT JOIN census.facts As F ON
T.tract_id = F.tract_id
GROUP BY T.tract_id;

```

9-4 9-7
 <http://explain.depesz.com>
 HTML

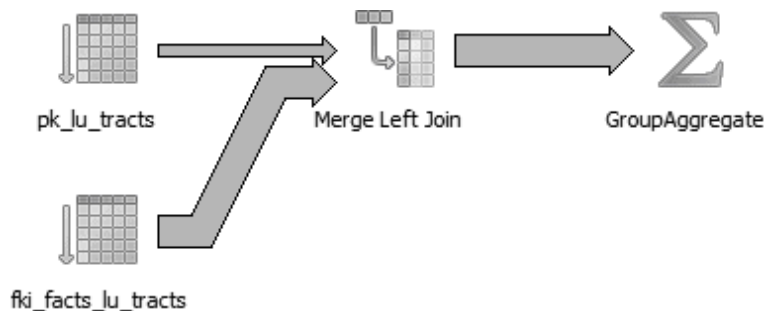


9-4 SQL

| HTML | TEXT | STATS | | | |
|-----------|-----------|--------|------|--------|--|
| exclusive | inclusive | rows x | rows | loops | node |
| 10.709 | 1292.135 | ↑ 1.0 | 1478 | 1 | → Seq Scan on lu_tracts t
(cost=0.00..615535.37 rows=1478 width=12) (actual time |
| | | | | | SubPlan (forSeq Scan) |
| 63.554 | 264.562 | ↑ 1.0 | 1 | 1478 | → Aggregate (cost=207.86..207.87 rows=1 width=0) (ac |
| 153.712 | 201.008 | ↑ 1.0 | 68 | 1478 | → Bitmap Heap Scan on facts f
(cost=4.79..207.69 rows=68 width=0) (actual time
Recheck Cond: ((tract_id)::text = (t.tract_id)::text) |
| 47.296 | 47.296 | ↑ 1.0 | 68 | 1478 | → Bitmap Index Scan on fki_facts_lu_tracts
(cost=0.00..4.78 rows=68 width=0) (actual time
Index Cond: ((tract_id)::text = (t.tract_id)::text) |
| 59.120 | 1016.864 | ↑ 1.0 | 1 | 1478 | → Aggregate (cost=208.56..208.57 rows=1 width=0) (ac |
| 314.814 | 957.744 | ↑ 1.0 | 68 | 1478 | → Nested Loop
(cost=207.86..208.39 rows=68 width=0) (actual time |
| 155.190 | 341.418 | ↓ 68.0 | 68 | 1478 | → HashAggregate
(cost=207.86..207.87 rows=1 width=4) (actual time |
| 141.888 | 186.228 | ↑ 1.0 | 68 | 1478 | → Bitmap Heap Scan on facts f
(cost=4.79..207.69 rows=68 width=4) (actual time
Recheck Cond: ((tract_id)::text = (t.tract_id)::text) |
| 44.340 | 44.340 | ↑ 1.0 | 68 | 1478 | → Bitmap Index Scan on fki_facts_lu_tracts
(cost=0.00..4.78 rows=68 width=0) (actual time
Index Cond: ((tract_id)::text = (t.tract_id)::text) |
| 301.512 | 301.512 | ↑ 1.0 | 1 | 100504 | → Index Scan using pk_lu_fact_types on lu_fact_types
(cost=0.00..0.50 rows=1 width=4) (actual time
Index Cond: (fact_type_id = f.fact_type_id) |

9-5 SQL

9-6 9-8



9-6

データベースのクエリは、SQL の構文規則に従って記述されます。データベースは、クエリを実行するために、データベースのスキーマとデータを参照します。

9.3.2 データベース SELECT * のクエリ

SELECT * は、データベースのすべてのデータを返すクエリです。10 行のデータを 1000 行のデータに変換するクエリを実行すると、データベースは、クエリを実行するために、データベースのスキーマとデータを参照します。

データベースのクエリは、PostgreSQL の TOAST (The Oversized-Attribute Storage Technique) を使用して実行されます。TOAST は、データベースのスキーマとデータを参照して、クエリを実行するために、データベースのスキーマとデータを参照します。TOAST は、データベースのスキーマとデータを参照して、クエリを実行するために、データベースのスキーマとデータを参照します。SELECT * は、データベースのすべてのデータを返すクエリです。

データベースのクエリは、PostgreSQL の TOAST (The Oversized-Attribute Storage Technique) を使用して実行されます。SELECT * は、データベースのすべてのデータを返すクエリです。データベースのクエリは、PostgreSQL の TOAST (The Oversized-Attribute Storage Technique) を使用して実行されます。SELECT * は、データベースのすべてのデータを返すクエリです。

データベースのクエリは、9-7 のクエリを実行するために、データベースのスキーマとデータを参照します。census データベースのクエリは、データベースのスキーマとデータを参照します。

```
CREATE OR REPLACE VIEW vw_stats AS
SELECT tract_id,
       (SELECT COUNT(*)
        FROM census.facts As F
        WHERE F.tract_id = T.tract_id) As num_facts,
       (SELECT COUNT(*)
        FROM census.lu_fact_types As Y
        WHERE Y.fact_type_id IN (
          SELECT fact_type_id
          FROM census.facts F
          WHERE F.tract_id = T.tract_id
        )
        ) As num_fact_types
FROM census.lu_tracts As T;
```

□□□□□□□□□□□□□□□□

```
SELECT tract_id FROM vw_stats;
```

□□□□□□□□□□□□□□□□□□ 21 □□□□□□□□□□□□□□□□ num_facts □
num_fact_type □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□ facts □□□□□□□□□□□□□□□□□□□□
□□□

```
SELECT * FROM vw_stats;
```

□□□□□□□□□□□□□□□□ 681 □□□□□□□□□□ 9-4 □□□□□□□□□□□□□□□□□□
□□□
□□□

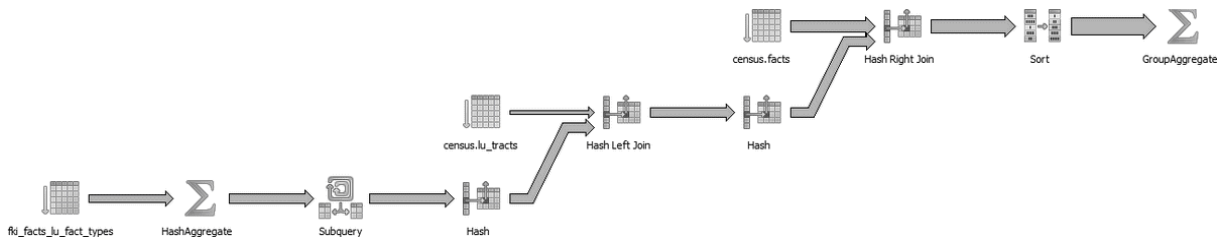
9.3.3 □□CASE □□

CASE □ ANSI SQL □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□ CASE □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
CASE □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 9-9 □□□□□□□□□□

□□ 9-9 □□□□□□□ CASE

```
SELECT T.tract_id, COUNT(*) As tot, type_1.tot AS type_1
FROM
  census.lu_tracts AS T LEFT JOIN
  (SELECT tract_id, COUNT(*) As tot
   FROM census.facts
   WHERE fact_type_id = 131
   GROUP BY tract_id
  ) As type_1 ON T.tract_id = type_1.tract_id LEFT JOIN
  census.facts AS F ON T.tract_id = F.tract_id
GROUP BY T.tract_id, type_1.tot;
```

□ 9-7 □□□ 9-9 □□□□□□□□□□



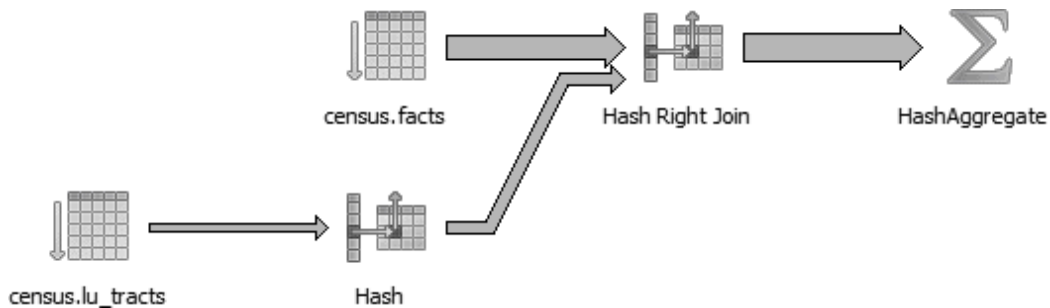
9-7 CASE

CASE 9-10

9-10 CASE

```
SELECT T.tract_id, COUNT(*) As tot,
       COUNT(CASE WHEN F.fact_type_id = 131 THEN 1 ELSE NULL END) AS
type_1
FROM census.lu_tracts AS T LEFT JOIN census.facts AS F
ON T.tract_id = F.tract_id
GROUP BY T.tract_id;
```

9-8 9-10



9-8 CASE

fact_type facts

9.3.4 Filter CASE

PostgreSQL 9.4 FILTER 7.15 CASE FILTER CASE

9-11 FILTER 9-10

9-11 FILTER

```
SELECT T.tract_id, COUNT(*) As tot,  
       COUNT(*) FILTER (WHERE F.fact_type_id = 131) AS type_1  
FROM census.lu_tracts AS T LEFT JOIN census.facts AS F  
ON T.tract_id = F.tract_id  
GROUP BY T.tract_id;
```

FILTER CASE 1

9.4

PostgreSQL CPU 50% 75%

PostgreSQL 9.6

PostgreSQL 10

- DML
- DDL
- COUNT SUM DISTINCT ORDER BY
- PARALLEL UNSAFE 8.1 PARALLEL

- `dynamic_shared_memory` 设置为 `none`
- `max_worker_processes` 设置为 0
- `max_parallel_workers` 在 PostgreSQL 10 中默认为 0，但应设置为与 `max_worker_processes` 相同的值
- `max_parallel_workers_per_gather` 默认为 0，应设置为与 `max_worker_processes` 相同的值。PostgreSQL 10 中默认为 1，应设置为与 `max_parallel_workers` 相同的值

9.4.1 并行查询

并行查询是指将一个大的查询分解成多个小的查询，每个小的查询由一个 worker 进程执行。在 PostgreSQL 中，并行查询是由一个 leader 进程和多个 worker 进程组成的。leader 进程负责协调 worker 进程的工作，并将查询结果汇总起来。worker 进程负责执行查询的每个子查询。在 PostgreSQL 10 中，并行查询的默认行为是关闭的，需要手动启用。可以通过设置 `force_parallel_mode` 为 `true` 来启用并行查询。此外，还可以通过设置 `max_parallel_workers` 和 `max_parallel_workers_per_gather` 来控制并行查询的规模。

在 PostgreSQL 10 中，可以通过设置 `force_parallel_mode` 为 `true` 来启用并行查询。这可以通过在 `postgresql.conf` 文件中添加以下配置来实现：

在 `postgresql.conf` 文件中，找到 `force_parallel_mode` 配置项，并将其值设置为 `true`。保存文件并重启 PostgreSQL 服务。

```
Set force_parallel_mode = true;
```

图 9-4 展示了并行查询的执行过程。图 9-12 展示了并行查询的执行结果。

图 9-12 使用 `EXPLAIN (ANALYZE)` 查看并行查询的执行计划


```

Gather
  (cost=1029.57..1051.65 rows=192 width=64)
  (actual time=12.881..13.947 rows=1 loops=1)
  Workers Planned: 1
  Workers Launched: 1
  Single Copy: true
  -> HashAggregate
    (cost=29.57..32.45 rows=192 width=64)
    (actual time=0.230..0.231 rows=1 loops=1)
    Group Key: "left"((tract_id)::text, 5)
    -> Bitmap Heap Scan on hisp_pop
      (cost=10.25..28.61 rows=192 width=36)
      (actual time=0.127..0.184 rows=204 loops=1)
      Recheck Cond:
        (((tract_id)::text >= '25025000000'::text) AND
         ((tract_id)::text <= '25025999999'::text))
      -> Bitmap Index Scan on hisp_pop_pkey
        (cost=0.00..10.20 rows=192 width=0)
        (actual time=0.106..0.106 rows=204 loops=1)
        Index Cond:
          (((tract_id)::text >= '25025000000'::text)
           AND
           ((tract_id)::text <= '25025999999'::text))
Planning time: 0.416 ms
Execution time: 16.160 ms

```

worker

worker

650

9-13 GROUP BY

```

set max_parallel_workers_per_gather=4;
EXPLAIN ANALYZE VERBOSE
SELECT COUNT(*), area_type_code
FROM labor
GROUP BY area_type_code
ORDER BY area_type_code;

```

```

Finalize GroupAggregate
  (cost=104596.49..104596.61 rows=3 width=10)
  (actual time=500.440..500.444 rows=3 loops=1)
  Output: COUNT(*), area_type_code
  Group Key: labor.area_type_code
  -> Sort
    (cost=104596.49..104596.52 rows=12 width=10)
    (actual time=500.433..500.435 rows=15 loops=1)
    Output: area_type_code, (PARTIAL COUNT(*))
    Sort Key: labor.area_type_code
    Sort Method: quicksort Memory: 25kB
    -> Gather
      (cost=104595.05..104596.28 rows=12 width=10)
      (actual time=500.159..500.382 rows=15 loops=1)
      Output: area_type_code, (PARTIAL COUNT(*))
      Workers Planned: 4
      Workers Launched: 4
      -> Partial HashAggregate
        (cost=103595.05..103595.08 rows=3 width=10)
        (actual time=483.081..483.082 rows=3 loops=5)
        Output: area_type_code, PARTIAL count(*)
        Group Key: labor.area_type_code
        Worker 0: actual time=476.705..476.706 rows=3
loops=1
        Worker 1: actual time=480.704..480.705 rows=3
loops=1
        Worker 2: actual time=480.598..480.599 rows=3
loops=1
        Worker 3: actual time=478.000..478.000 rows=3
loops=1
        -> Parallel Seq Scan on public.labor
          (cost=0.00..95516.70 rows=1615670 width=2)
          (actual time=1.550..282.833 rows=1292543
loops=5)
          Output: area_type_code
          Worker 0: actual time=0.078..282.698
rows=1278313 loops=1
          Worker 1: actual time=3.497..282.068
rows=1338095 loops=1
          Worker 2: actual time=3.378..281.273
rows=1232359 loops=1
          Worker 3: actual time=0.761..278.013
rows=1318569 loops=1
Planning time: 0.060 ms
Execution time: 512.667 ms

```

max_parallel_workers_per_gather=0 9-14

9-14 GROUP BY

```
set max_parallel_workers_per_gather=0;
EXPLAIN ANALYZE VERBOSE
SELECT COUNT(*), area_type_code
FROM labor
GROUP BY area_type_code
ORDER BY area_type_code;

Sort
  (cost=176300.24..176300.25 rows=3 width=10)
  (actual time=1647.060..1647.060 rows=3 loops=1)
  Output: (COUNT(*)), area_type_code
  Sort Key: labor.area_type_code
  Sort Method: quicksort Memory: 25kB
  -> HashAggregate
    (cost=176300.19..176300.22 rows=3 width=10)
    (actual time=1647.025..1647.025 rows=3 loops=1)
    Output: count(*), area_type_code
    Group Key: labor.area_type_code
    -> Seq Scan on public.labor
      (cost=0.00..143986.79 rows=6462679 width=2)
      (actual time=0.076..620.563 rows=6462713 loops=1)
      Output: series_id, year, period, value, footnote_codes,
area_type_code
Planning time: 0.054 ms
Execution time: 1647.115 ms
```

| count | area_type_code |
|---------|----------------|
| 3718937 | M |
| 2105205 | N |
| 638571 | S |

(3 rows)

worker worker 280

9.4.2 扫描器

扫描器是负责扫描数据块的 worker。PostgreSQL 9.6 扫描器使用堆扫描 (heap scan) 或索引扫描 (index scan)¹。index-only scan 扫描器使用 B-树索引扫描器。B-树索引扫描器使用堆扫描器扫描数据块。worker 扫描器扫描数据块²。

¹ 扫描器使用堆扫描或索引扫描。——扫描器

² 扫描器使用堆扫描或索引扫描。PAGE 扫描器扫描数据块。BIT 扫描器扫描数据块。PAGE 扫描器扫描数据块。PAGE 扫描器扫描数据块。BIT 扫描器扫描数据块。1 扫描器扫描数据块。IO 扫描器扫描数据块。 <https://dba.stackexchange.com/questions/119386/understanding-bitmap-heap-scan-and-bitmap-index-scan> ——扫描器

9.4.3 扫描器

PostgreSQL 9.6 扫描器使用堆扫描 (heap scan) 或索引扫描 (index scan)。

worker 扫描器扫描数据块。worker 扫描器扫描数据块。

worker 扫描器扫描数据块。worker 扫描器扫描数据块。worker 扫描器扫描数据块。worker 扫描器扫描数据块。

PostgreSQL 10 扫描器使用 merge 扫描器。merge 扫描器扫描数据块。worker 扫描器扫描数据块。worker 扫描器扫描数据块。

9.5 扫描器

扫描器是负责扫描数据块的 worker。扫描器是负责扫描数据块的 worker。扫描器是负责扫描数据块的 worker。扫描器是负责扫描数据块的 worker。

11

9.5.1 〇〇〇〇

```

PostgreSQL
PostgreSQL " "

```

```

enable_nestloop enable_seqscan
" "

```

9.5.2 □□□□□□□□

```

pg_stat_user_indexes
pg_stat_user_tables
pg_stat_statements
9.2

```

```

##### 7-22 ##### fact_subcats
##### GIN ##### GIN #####

```

```
CREATE INDEX idx_lu_fact_types ON census.lu_fact_types USING gin
(fact subcats);
```

fact_subcats
"White alone" "Asian alone"
9-15

9-15

```
set enable_seqscan = true;
EXPLAIN (ANALYZE)
SELECT *
FROM census.lu_fact_types
WHERE fact_subcats && '{White alone, Black alone}'::varchar[];

Seq Scan on lu_fact_types
(cost=0.00..2.85 rows=2 width=200)
(actual time=0.066..0.076 rows=2 loops=1)
  Filter: (fact_subcats
&& '{"White alone","Black alone"}'::character varying[])
    Rows Removed by Filter: 66
Planning time: 0.182 ms
Execution time: 0.108 ms
```

9-16

9-16

```
set enable_seqscan = false;
EXPLAIN (ANALYZE)
SELECT *
FROM census.lu_fact_types
WHERE fact_subcats && '{White alone, Black alone}'::varchar[];

Bitmap Heap Scan on lu_fact_types
(cost=12.02..14.04 rows=2 width=200)
(actual time=0.058..0.058 rows=2 loops=1)
  Recheck Cond: (fact_subcats
&& '{"White alone","Black alone"}'::character varying[])
    Heap Blocks: exact=1
      -> Bitmap Index Scan on idx_lu_fact_types
        (cost=0.00..12.02 rows=2 width=0)
        (actual time=0.048..0.048 rows=2 loops=1)
        Index Cond: (fact_subcats
&& '{"White alone","Black alone"}'::character varying[])
```

```
Planning time: 0.230 ms
Execution time: 0.119 ms
```

이 쿼리는 census.lu_fact_types 테이블에서 'White alone' = ANY(fact_subcats)인 모든 행을 반환합니다. 이 쿼리는 매우 느리게 실행됩니다.

이 쿼리는 census.lu_fact_types 테이블에서 'White alone' = ANY(fact_subcats)인 모든 행을 반환합니다.

```
SELECT * FROM census.lu_fact_types WHERE 'White alone' =
ANY(fact_subcats);
```

enable_seqscan 옵션을 사용하여 SQL 쿼리를 실행하면, 이 쿼리는 훨씬 더 빠르게 실행됩니다.

9.5.3 인덱스

이 쿼리는 census.lu_fact_types 테이블에서 'White alone' = ANY(fact_subcats)인 모든 행을 반환합니다. 이 쿼리는 매우 느리게 실행됩니다.

이 쿼리는 census.lu_fact_types 테이블에서 'White alone' = ANY(fact_subcats)인 모든 행을 반환합니다. 이 쿼리는 매우 느리게 실행됩니다. 20% 인덱스 사용률로 STATISTICS 옵션을 사용하여 쿼리를 실행합니다.

pg_stats 뷰를 사용하여 쿼리를 실행합니다. 9-17 참조

9-17 인덱스

```
SELECT
    attname As colname,
    n_distinct,
    most_common_vals AS common_vals,
```

```

    most_common_freqs As dist_freq
FROM pg_stats
WHERE tablename = 'facts'
ORDER BY schemaname, tablename, attname;

```

| colname | n_distinct | common_vals | dist_freq |
|--------------|------------|-------------------|---------------------|
| fact_type_id | 68 | {135,113... | |
| perc | 985 | {0.00,... | |
| tract_id | 1478 | {25025090300... | |
| val | 3391 | {0.000,1.000,2... | |
| yr | 2 | {2011,2010} | {0.748933,0.251067} |

pg_stats 表包含统计信息。您可以通过 VACUUM ANALYZE 命令来更新统计信息。

您可以通过 WHERE 子句来指定要更新的表。

```

ALTER TABLE census.facts ALTER COLUMN fact_type_id SET STATISTICS
1000;

```

PostgreSQL 10 引入了 CREATE STATISTICS 命令，用于创建统计信息。您可以通过该命令来指定要创建的统计信息的名称、依赖项以及要使用的表。

9-18 创建统计信息

```

CREATE STATISTICS census.stats_facts_type_yr_dep_dist
(dependencies, ndistinct)
ON fact_type_id, yr FROM census.facts;

```



```
ANALYZE census.facts;
```

```
CREATE STATISTICS census_stats_facts_type_yr_dep_dist ON
census.facts USING fact_type_id yr USING census.schema
census.stats_facts_type_yr_dep_dist ON census.schema
schema USING schema USING schema USING schema
```

CREATE STATISTICS census_stats_facts_type_yr_dep_dist ON

- CREATE STATISTICS census_stats_facts_type_yr_dep_dist ON
02109 USING fact_type_id yr USING census.schema
city = 'Boston' and zip = '02109' USING
- CREATE ndistinct census_stats_facts_type_yr_dep_dist ON
ndistinct USING GROUP BY USING GROUP BY

```
CREATE STATISTICS pg_statistic_ext ON
DROP STATISTICS pg_statistic_ext ANALYZE
ANALYZE pg_statistic_ext
autovacuum pg_statistic_ext ANALYZE
```

9.5.4 CREATE STATISTICS

CREATE STATISTICS random_page_cost ON
RPC USING random_page_cost
RPC USING 4
SAN

database USING RPC
postgresql.conf
RPC

```
ALTER TABLESPACE pg_default SET (random_page_cost=2);
```

“Random Page Cost Revisited”

- NAS/SAN 2.5 3.0
- EBS 1 Heroku 2.0
- iSCSI SAN 6.0
- 2.0 2.5
- NvRAM NAND 1.5

9.6

CTE

pg_buffercache

```
CREATE EXTENSION pg_buffercache;
```

pg_buffercache 9-19

9-19

```
SELECT
    C.relname,
    COUNT(CASE WHEN B.isdirty THEN 1 ELSE NULL END) As
dirty_buffers,
    COUNT(*) As num_buffers
FROM
    pg_class AS C INNER JOIN
    pg_buffercache B ON C.relfilenode = B.relfilenode INNER JOIN
    pg_database D ON B.reldatabase = D.oid AND D.datname =
current_database()
WHERE C.relname IN ('facts','lu_fact_types')
```

```
GROUP BY C.relname;
```

9-19 facts lu_fact_types SQL 9-19

```
SELECT T.fact_subcats[2], COUNT(*) As num_fact
FROM
    census.facts As F
    INNER JOIN
    census.lu_fact_types AS T ON F.fact_type_id =
T.fact_type_id
GROUP BY T.fact_subcats[2];
```

10% 9-19

| relname | dirty_buffers | num_buffers |
|---------------|---------------|-------------|
| facts | 0 | 736 |
| lu_fact_types | 0 | 4 |

postgresql.conf shared_buffers

pg_prewarm

10

```
PostgreSQL
PostgreSQL
foreign data wrapper
FDW
9.3
postgres_fdw
hadoop_fdw
ogr_fdw
10.3.4
```

10.1

TB

[illegible]

10.1.1 □□□□□□□□

□ □

□ □ □ □

PostgreSQL
 publisher PostgreSQL 10
 publisher/subscriber /

□□□□□

[illegible]

write-ahead log (WAL)

WAL 是 PostgreSQL 数据库系统的一个核心组件，它负责记录数据库的所有更改。在 PostgreSQL 9.6 之前，WAL 是可选的，而在 9.6 及以后版本中，WAL 是强制性的。WAL 的主要作用是确保数据库的持久性和一致性。

简介

PostgreSQL 数据库系统使用 WAL 来记录所有的更改。在 PostgreSQL 9.6 之前，WAL 是可选的，而在 9.6 及以后版本中，WAL 是强制性的。WAL 的主要作用是确保数据库的持久性和一致性。在 PostgreSQL 10 中，WAL 的默认配置是 `synchronous_standby_names` 设置为 `FIRST`，这意味着在提交事务之前，必须等待至少一个 standby 节点完成 WAL 的复制。在 PostgreSQL 9.6 中，WAL 的默认配置是 `synchronous_standby_names` 设置为 `ANY`，这意味着在提交事务之前，只需要等待至少一个 standby 节点完成 WAL 的复制。

配置

WAL 的配置可以通过修改 `postgresql.conf` 文件来实现。在 `postgresql.conf` 文件中，`synchronous_standby_names` 参数的配置决定了在提交事务之前，必须等待多少个 standby 节点完成 WAL 的复制。在 PostgreSQL 9.6 中，`synchronous_standby_names` 的默认配置是 `ANY`，而在 PostgreSQL 10 中，默认配置是 `FIRST`。

PostgreSQL 9.4 引入了 `replication slot` 的概念，它用于跟踪 WAL 的复制进度。在 PostgreSQL 9.4 中，`replication slot` 的默认配置是 `ANY`，而在 PostgreSQL 10 中，默认配置是 `FIRST`。在 PostgreSQL 9.4 中，`replication slot` 的默认配置是 `ANY`，而在 PostgreSQL 10 中，默认配置是 `FIRST`。

总结


```

CREATE EXTENSION

```

10.1.2 □□□□□□

PostgreSQL WAL
CPU 32/64
PostgreSQL PostgreSQL
PostgreSQL PostgreSQL
PostgreSQL PostgreSQL

データベース PostgreSQL

- 9.4 物理备份和逻辑备份“区别”和优缺点
- 9.5 物理备份和逻辑备份“区别”和优缺点
- 9.6 物理备份和逻辑备份“区别”和优缺点
- PostgreSQL 10 物理备份和逻辑备份“区别”和优缺点

PostgreSQL 10 から PostgreSQL 9.4 へ
 pglogical をインストールする
 PostgreSQL から PostgreSQL 10 へ PostgreSQL
 9.4~9.6 から PostgreSQL を pglogical
 PostgreSQL 10 から pglogical

10.1.3

データベースの運用において、データベースの構造を変更する操作は、データベースの運用に大きな影響を与える可能性があります。DDL (Data Definition Language) の操作は、データベースの構造を変更する操作であり、データベースの運用に大きな影響を与える可能性があります。

データベースの運用において、データベースの複製、クラスタリング、および接続プーリングは、データベースの運用に大きな影響を与える可能性があります。

10.2 データベースの複製

データベースの複製は、PostgreSQL のデータベースの複製を行うための方法です。データベースの複製は、データベースの運用に大きな影響を与える可能性があります。WAL (Write Ahead Log) は、データベースの複製を行うための方法です。

10.2.1 データベースの複製

データベースの複製を行うためには、データベースの複製を行う必要があります。

(1) データベースの複製を行うための操作

```
CREATE ROLE pgrepuser REPLICATION LOGIN PASSWORD 'woohoo';
```

(2) postgresql.conf の設定を変更する操作
ALTER SYSTEM set wal_level = logical; SELECT pg_reload_conf();

```
listen_addresses = *  
wal_level = hot_standby  
archive_mode = on  
max_wal_senders = 5  
wal_keep_segments = 10
```

データベースの複製を行うためには、wal_level を logical に設定する必要があります。logical は、hot_standby をサポートするための設定です。logical は、PostgreSQL のデータベースの複製を行うための方法です。

PostgreSQL のデータベースの複製を行うためには、wal_keep_segments を設定する必要があります。

PostgreSQL 9.6 wal_level replica
hot_standby PostgreSQL 9.6
hot_standby replica

(3) postgresql.conf archive_command ALTER
SYSTEM WAL PostgreSQL
PostgreSQL “PostgreSQL
PGStandby”

Linux/Unix archive_command

```
archive_command = 'cp %p ../archive/%f'
```

rsync cp

```
archive_command = 'rsync -av %p postgres@192.168.0.10:archive/%f'
```

Windows

```
archive_command = 'copy %p ..\\archive\\%f'
```

(4) pg_hba.conf pgrepuser
PostgreSQL IP 192.168.0.1
192.168.0.254 MD5

```
host replication pgrepuser 192.168.0.0/24 md5
```

(5) PostgreSQL

PostgreSQL bin pg_basebackup
PostgreSQL

pg_basebackup 使用 --xlog-method=stream 使用 WAL 复制方法复制 WAL 记录 -R 设置复制槽



PostgreSQL 10 使用 pg_xlog 复制槽 pg_wal

复制槽名称为 192.168.0.1

```
pg_basebackup -D /target_dir -h 192.168.0.1 \
--port=5432 --checkpoint=fast
--xlog-method=stream -R
```

pg_basebackup 使用 TAR 格式复制槽 tar.gz 使用 --xlog-method 复制槽名称

```
pg_basebackup -Z9 -D /target_dir/ -h 192.168.0.1 -Ft -Xfetch
```

WAL 复制槽 PostgreSQL 10 使用 pg_receivexlog 复制槽 PostgreSQL 10 使用 pg_receivewal 复制槽

10.2.2 复制槽

复制槽使用 CREATE EXTENSION 创建 WAL 复制槽

(1) 复制槽 PostgreSQL 复制槽 PostgreSQL 复制槽

(2) PostgreSQL 설치

(3) pg_basebackup 실행

(4) postgresql.auto.conf 수정

```
hot_standby = on  
max_connections = 20 #set to higher or equal to master
```

(5) postgresql.auto.conf 및 postgresql.conf 수정
PGPORT

(6) data recovery.conf
IP pg_basebackup
trigger_file

```
standby_mode = 'on'  
primary_conninfo = 'host=192.168.0.1 port=5432 user=pgrepuser  
password=woohoo application_name=replica1'  
trigger_file = 'failover.now'
```

(7) recovery.conf
Linux/Unix

```
restore_command = 'cp %p ../archive/%f'
```

Windows

```
restore_command = 'copy %p ..\archive\%f'
```

archive

10.2.3

pg_basebackup recovery.conf

PostgreSQL data failover.now recovery.conf recovery.done PostgreSQL

10.2.4 database

PostgreSQL 10 PostgreSQL database PostgreSQL 10 Linux PostgreSQL Windows PostgreSQL

“publisher” subscriber database CREATE PUBLICATION database CREATE SUBSCRIPTION DDL

PostgreSQL 10 PostgreSQL 5447 PostgreSQL 5448 PostgreSQL

(1) PostgreSQL

```
SHOW wal_level
```

将 wal_level 设置为 logical

```
ALTER SYSTEM SET wal_level = logical;
```

将 PostgreSQL 升级到 12

在升级之前，需要先将 wal_level 设置为 logical

(2) 将 postgresql_book database 导出为 pg_dump 格式
pg_dump postgresql_book > postgresql_book.sql

```
pg_dump -U postgres -p5447 -Fp --section pre-data --section post-data \
-f pub_struct.sql postgresql_book
```

在 postgresql 12 中创建数据库

```
CREATE DATABASE book_sub;
\connect book_sub;
\i pub_struct.sql
```

(3) 将 postgresql_book database 导出为 pg_dump 格式
CREATE PUBLICATION book_sub_pub FOR ALL TABLES;

```
CREATE PUBLICATION full_db_pub
FOR ALL TABLES;
```

(4) 将 postgresql_book database 导出为 pg_dump 格式
pg_dump postgresql_book > postgresql_book.sql

10.3.1 安装文件扩展

安装 file_fdw 扩展 FDW 扩展是 PostgreSQL 9.5 引入的，用于从外部数据源加载数据到 SQL 数据库。

```
CREATE EXTENSION file_fdw;
```

安装 file_fdw 扩展后，需要创建一个 FDW 扩展，用于指定数据源。以下是一个示例，用于指定一个名为 my_server 的 FDW 扩展。

```
CREATE SERVER my_server FOREIGN DATA WRAPPER file_fdw;
```

安装完 file_fdw 扩展后，需要创建一个 schema 来存放从外部数据源加载的数据。以下是一个示例，用于创建一个名为 staging 的 schema，并设置其所有者为 10-1。

安装完 file_fdw 扩展后，需要创建一个 schema 来存放从外部数据源加载的数据。以下是一个示例，用于创建一个名为 staging 的 schema，并设置其所有者为 10-1。

```
Dev|Company  
Tom Lane|Crunchy Data  
Bruce Momjian|EnterpriseDB
```

图 10-1 安装文件扩展

```
CREATE FOREIGN TABLE staging.devs (developer VARCHAR(150), company  
VARCHAR(150))  
SERVER my_server  
OPTIONS (  
    format 'csv',  
    header 'true',  
    filename '/postgresql_book/ch10/devs.psv',  
    delimiter '|',  
    null''  
);
```

CSV 格式，即“csv”格式，
CSV 是 comma seperated values 的缩写，即逗号分隔值。
CSV 格式 FDW 是 CSV 格式的外数据源。
CSV 格式的外数据源。

SQL 语句如下：

```
SELECT * FROM staging.devs WHERE developer LIKE 'T%';
```

删除外数据源：

```
DROP FOREIGN TABLE staging.devs;
```

10.3.2 安装 file_textarray_fdw

file_textarray_fdw 是 FDW 的一种，它可以将文本文件中的数据加载到 PostgreSQL 的文本数组（text[]）类型中。

file_textarray_fdw 是 PostgreSQL 的一个扩展，它可以将文本文件中的数据加载到 PostgreSQL 的文本数组（text[]）类型中。Adunstan GitHub 上有一个 file_textarray_fdw 的仓库，它提供了 PostgreSQL 9.4 及以上版本的安装指南。PostgreSQL 9.4 及以上版本支持 file_textarray_fdw 扩展。FDW 是 Foreign Data Wrapper 的缩写。

Linux/Unix 系统安装 postgresql-dev 包。Windows 系统安装 PostgreSQL 9.4 及以上版本。Windows 32/64 位 PostgreSQL 9.4。

<http://www.postgresql.com/journal/archives/340-Foreign-Data-Wrappers-for-PostgreSQL-9.4-Windows.html>

<http://bit.ly/2oRDY6X> Windows 32/64 位

PostgreSQL 9.5 或 PostgreSQL 9.6

<http://www.postgresql.com/journal/archives/361-Foreign-Data-Wrappers-for-PostgreSQL-9.5-windows.html>

FDW 拡張機能のインストール

```
CREATE EXTENSION file_textarray_fdw;
```

外部データラッパーとして FDW をインストール

```
CREATE SERVER file_taserver FOREIGN DATA WRAPPER  
file_textarray_fdw;
```

外部データラッパーとしてインストールした schema 10-2 の外部データラッパー
staging schema

図 10-2 外部データラッパーのインストール

```
CREATE FOREIGN TABLE staging.factfinder_array (x text[])  
SERVER file_taserver  
OPTIONS (  
    format 'csv',  
    filename '/postgresql_book/ch10/DEC_10_SF1_QTH1_with_ann.csv',  
    header 'false',  
    delimiter ',',  
    quote '',  
    encoding 'latin1',  
    null ''  
);
```

外部データラッパー CSV をインストールした 8 の外部データラッパーのインストール
staging schema の factfinder_array テーブルの factfinder_array テーブルの factfinder_array
GEO.id

```
SELECT unnest(x) FROM staging.factfinder_array WHERE x[1] =  
'GEO.id'
```

外部データラッパーのインストール

```
SELECT x[1] As geo_id, x[2] As tract_id  
FROM staging.factfinder_array WHERE x[1] ~ '[0-9]+';
```

10.3.3 安装PostgreSQL扩展

在 9.3 版本之前 PostgreSQL 扩展 postgres_fdw 是 FDW 扩展，PostgreSQL 扩展 postgres_fdw 是 PostgreSQL 扩展。

安装 FDW 扩展

```
CREATE EXTENSION postgres_fdw;
```

安装 server

```
CREATE SERVER book_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'localhost', port '5432', dbname 'postgresql_book');
```

安装 server 选项 ALTER SERVER

```
ALTER SERVER book_server OPTIONS (SET host 'prod');
```



安装 database 扩展

安装 ¹ public 扩展

¹ 在“安装”扩展时，需要指定扩展的名称和版本。

```
CREATE USER MAPPING FOR public SERVER book_server
OPTIONS (user 'role_on_foreign', password 'your_password');
```


FOREIGN SCHEMA 옵션은 FDW 옵션으로만 사용할 수 있습니다.
postgres_fdw 옵션으로만 사용할 수 있습니다.

import_collate

PostgreSQL 옵션으로만 사용할 수 있습니다. true

import_default

PostgreSQL 옵션으로만 사용할 수 있습니다. false
옵션으로만 사용할 수 있습니다.

Insert 옵션으로만 사용할 수 있습니다.
옵션으로만 사용할 수 있습니다.
옵션으로만 사용할 수 있습니다.

import_not_null

PostgreSQL 옵션으로만 NOT NULL 옵션으로만 사용할 수 있습니다.
true

10-4 PostgreSQL books.public schema
옵션으로만 사용할 수 있습니다.

10-4 IMPORT FOREIGN SCHEMA 옵션으로만 schema
옵션으로만 사용할 수 있습니다.

```
CREATE SCHEMA remote_census;  
IMPORT FOREIGN SCHEMA public  
FROM SERVER book_server  
INTO remote_census  
OPTIONS (import_default 'true');
```

10-4 IMPORT FOREIGN SCHEMA 옵션으로만 schema 옵션으로
remote_census schema 옵션으로만 사용할 수 있습니다.

옵션으로만 사용할 수 있습니다. LIMIT TO EXCEPT 옵션으로만 사용할 수 있습니다. facts
lu_fact_type 옵션으로만 사용할 수 있습니다.

ServerOracle 数据库数据库数据库 PostGIS 数据库 PostgreSQL 数据库

PostGIS 数据库数据库数据库 ogr_fdw 数据库数据库EnterpriseDB 数据库 StackBuilder 数据库数据库 Windows 数据库 PostGIS 数据库数据库 ogr_fdw 数据库 CentOS 数据库 Linux 数据库RHEL数据库数据库 yum 数据库 数据库 yum. postgresql.org 数据库数据库 ogr_fdw 数据库 BigSQL 数据库 数据库 PostgreSQL 数据库数据库数据库 ogr_fdw 数据库数据库数据库数据库 ogr_fdw 数据库 GitHub 数据库数据库

数据库数据库数据库ogr_fdw 数据库数据库“数据库数据库数据库”数据库Geospatial Data Abstraction Library数据库数据库数据库数据库数据库数据库数据库 ogr_fdw 数据库数据库数据库数据库 GDAL 数据库GDAL 数据库数据库数据库数据库数据库数据库数据库数据库数据库数据库数据库 GDAL 数据库 GDAL 数据库数据库数据库GDAL 数据库 PostgreSQL 数据库数据库数据库数据库数据库 GDAL 数据库数据库数据库数据库数据库数据库数据库 PostgreSQL

GDAL 数据库数据库数据库数据库 Excel 数据库LibreOffice Calc 数据库ODBC 数据库 数据库 Web 数据库数据库数据库 Windows 数据库数据库 Microsoft Access 数据库数据库 Linux/Mac 数据库数据库数据库

数据库 ogr_fdw 数据库数据库数据库数据库 PostgreSQL 数据库 ogr_fdw 数据库 database数据库数据库数据库数据库

```
CREATE EXTENSION ogr_fdw;
```

数据库 ogr_fdw 数据库数据库数据库数据库数据库数据库数据库数据库数据库数据库数据库数据库数据库数据库数据库 CSV 数据库CSV 数据库数据库数据库数据库数据库数据库数据库数据库数据库 CSV 数据库 数据库数据库数据库 Microsoft Excel 数据库 LibreOffice Calc 数据库数据库数据库数据库 数据库数据库数据库数据库数据库数据库数据库数据库数据库数据库 SQLite 数据库数据库 database 数据库数据库数据库数据库数据库数据库数据库数据库数据库数据库

数据库数据库数据库数据库 LibreOffice 数据库数据库数据库数据库数据库数据库数据库数据库数据库数据库 数据库

```
CREATE SERVER ogr_fdw_wb
FOREIGN DATA WRAPPER ogr_fdw
```

```
CREATE SCHEMA wb_data;
IMPORT FOREIGN SCHEMA ogr_all
FROM SERVER ogr_fdw_wb INTO wb_data;
```

```
CREATE SCHEMA wb_data;  
IMPORT FOREIGN SCHEMA "Finance"  
FROM SERVER ogr_fdw wb INTO wb_data;
```

```
CREATE SERVER ogr_fdw_ff
FOREIGN DATA WRAPPER ogr_fdw
OPTIONS (datasource '/fdw_data/factfinder', format 'CSV');
CREATE SCHEMA ff;
IMPORT FOREIGN SCHEMA "Housing"
FROM SERVER ogr_fdw_ff INTO ff;
```

```

##### Housing_2015.csv ##### Housing_2016.csv #####
##### schema ff ##### housing_2015 #####
housing 2016 #####

```

ogr_fdw 是一个用于从 ogr_fdw 服务器导入和导出数据的工具。它使用 IMPORT FOREIGN SCHEMA 语句来导入数据。

```
IMPORT FOREIGN SCHEMA "Housing"
  FROM SERVER ogr_fdw_ff INTO ff
  OPTIONS(launder_table_names 'false', launder_column_names
    'false');
```

ogr_fdw 支持从 ogr_fdw 服务器导入和导出数据。它支持从 ogr_fdw 服务器导入和导出数据。

10.3.5 安装和配置

ogr_fdw 是一个用于从 ogr_fdw 服务器导入和导出数据的工具。它使用 IMPORT FOREIGN SCHEMA 语句来导入数据。ogr_fdw 支持从 ogr_fdw 服务器导入和导出数据。它支持从 ogr_fdw 服务器导入和导出数据。

ogr_fdw 是一个用于从 ogr_fdw 服务器导入和导出数据的工具。它使用 IMPORT FOREIGN SCHEMA 语句来导入数据。ogr_fdw 支持从 ogr_fdw 服务器导入和导出数据。它支持从 ogr_fdw 服务器导入和导出数据。

PostgreSQL 支持从 ogr_fdw 服务器导入和导出数据。它支持从 ogr_fdw 服务器导入和导出数据。Linux/Unix 用户可以使用 postgresql-dev 包来安装。Windows 用户可以使用 Windows-32 9.1 和 Windows-64 9.3 版本。

Windows-32 9.1

http://www.postgresql.org/download/fdw_win32_91_bin.zip

Windows-64 9.3

http://www.postgresql.org/download/fdw_win64_93_bin.zip

ogr_fdw 是一个用于从 ogr_fdw 服务器导入和导出数据的工具。它使用 IMPORT FOREIGN SCHEMA 语句来导入数据。


```
GRANT SELECT ON TABLE www_fdw_google_search TO public;
```

データベース検索エンジン New in PostgreSQL 10 データベース検索エンジン
データベース検索エンジン HTML データベース検索エンジン

```
SELECT regexp_replace(title,E'(?x)(< [^>]*? >)',',','g') As title  
FROM www_fdw_google_search  
WHERE q= 'New in PostgreSQL 10'  
LIMIT 2;
```

データベース検索エンジン

```
title  
-----  
PostgreSQL 10 Roadmap  
PostgreSQL: Roadmap  
(2 rows)
```

PostgreSQL

A.1 WindowsLinux

EnterpriseDB Windows Linux データベース検索エンジン OS
データベース 32 64

データベース検索エンジン pgAdmin データベース検索エンジン
StackBuilder StackBuilder PostgreSQL データベース検索エンジン
JDBC .NET Ruby PostGIS phpPgAdmin データベース
pgAgent データベース検索エンジン

EnterpriseDB PostgreSQL データベース検索エンジン
Advanced Plus Oracle データベース検索エンジン

Postgres Plus Advanced Server

BigSQL PostgreSQL OpenSCG
BigSQL EnterpriseDB WindowsLinux
Mac 64

BigSQL EnterpriseDB
PostgreSQL DevOps
pgTSQL Microsoft SQL
Server T-SQL
pgBadger

PostGIS ogr_fdw hadoop_fdw
cassandra_fdw oracle_fdw

EnterpriseDB BigSQL Web
pgc pgc “pretty
good command-line”pgc
Linux yumapt-get
Windows
BigSQL

```
pgc update
pgc list
```

| Category | Component | Version | ReleaseDt | Status |
|-----------------|-----------|----------|------------|--------|
| PostgreSQL
1 | pg92 | 9.2.21-1 | 2017-05-11 | |
| PostgreSQL
1 | pg93 | 9.3.17-1 | 2017-05-11 | |
| PostgreSQL
1 | pg94 | 9.4.12-1 | 2017-05-11 | |
| PostgreSQL | pg95 | 9.5.7-1 | 2017-05-11 | |

| | | | | | |
|--------------|---------------------|----------|------------|-----------|--|
| 1 | | | | | |
| PostgreSQL | pg96 | 9.6.3-1 | 2017-05-11 | Installed | |
| 1 | | | | | |
| Extensions | cassandra_fdw3-pg96 | 3.0.1-1 | 2016-11-08 | | |
| 1 | | | | | |
| Extensions | hadoop_fdw2-pg96 | 2.5.0-1 | 2016-09-01 | | |
| 1 | | | | | |
| Extensions | oracle_fdw1-pg96 | 1.5.0-1 | 2016-09-01 | | |
| 1 | | | | | |
| Extensions | orafce3-pg96 | 3.3.1-1 | 2016-09-23 | | |
| 1 | | | | | |
| Extensions | pgaudit11-pg96 | 1.1.0-2 | 2017-05-18 | | |
| 1 | | | | | |
| Extensions | pgpartman2-pg96 | 2.6.4-1 | 2017-04-15 | | |
| 1 | | | | | |
| Extensions | pldebugger96-pg96 | 9.6.0-1 | 2016-12-28 | | |
| 1 | | | | | |
| Extensions | plprofiler3-pg96 | 3.2-1 | 2017-04-15 | | |
| 1 | | | | | |
| Extensions | postgis23-pg96 | 2.3.2-3 | 2017-05-18 | Installed | |
| 1 | | | | | |
| Extensions | setuser1-pg96 | 1.2.0-1 | 2017-02-23 | | |
| 1 | | | | | |
| Extensions | tds_fdw1-pg96 | 1.0.8-1 | 2016-11-23 | | |
| 1 | | | | | |
| Servers | pgdevops | 1.4-1 | 2017-05-18 | Installed | |
| 1 | | | | | |
| Applications | backrest | 1.18 | 2017-05-18 | | |
| 1 | | | | | |
| Applications | ora2pg | 18.1 | 2017-03-23 | | |
| 1 | | | | | |
| Applications | pgadmin3 | 1.23.0a | 2016-10-20 | Installed | |
| 1 | | | | | |
| Applications | pgagent | 3.4.1-1 | 2017-02-23 | | |
| 1 | | | | | |
| Applications | pgbadger | 9.1 | 2017-02-09 | | |
| 1 | | | | | |
| Frameworks | java8 | 8u121 | 2017-02-09 | | |
| 1 | | | | | |
| Frameworks | perl5 | 5.20.3.3 | 2016-03-14 | | |
| 1 | | | | | |
| Frameworks | python2 | 2.7.12-1 | 2016-10-20 | Installed | |
| 0 | | | | | |
| Frameworks | tcl86 | 8.6.4-1 | 2016-03-11 | | |
| 1 | | | | | |

pgAdmin4

```
pgc install pgdevops
```

pgDevops 提供了 Web 管理界面 pgAdmin4 以及 BigSQL 管理界面

初始化与启动

```
pgc init pgdevops
pgc start pgdevops
```

访问地址为 <http://localhost:8051>

升级操作 `pgc upgrade` 或 `pgc install`



除了通过二进制安装 PostgreSQL 之外，还可以通过 USB 安装 PostgreSQL EnterpriseDB 或 BigSQL 管理界面。EnterpriseDB 提供了“Starting PostgreSQL in Windows without Install”指南。

A.2 CentOS/Fedora/Red Hat/Scientific Linux

在 Linux/Unix 系统上安装 PostgreSQL 有多种方法，¹ 本文介绍通过 Yum 安装 PostgreSQL。

¹ “安装”指 backport——即通过社区或商业发行版提供——或直接从源代码编译。本文介绍的是通过 Yum 安装 PostgreSQL。

在 Linux 系统上安装 PostgreSQL 有多种方法，本文介绍通过 Yum 安装 PostgreSQL。

PostgreSQL Yum 2~4 PostgreSQL CentOS RedHat EL Fedora Scientific Linux Amazon AMI Oracle Enterprise

PostgreSQL PostgreSQL Yum PostgreSQL Yum PostgreSQL Online Yum

A.3 Debian Ubuntu

Debian Ubuntu apt-postgresql PostgreSQL apt-postgresql PostgreSQL yum postgresql Ubuntu Debian PostgreSQL

```
sudo apt-get install postgresql-9.6
```

postgresql-server-dev postgresql-server-dev

```
sudo apt-get install postgresql-server-dev-9.6
```

PostgreSQL Apt PostgreSQL packages PostgreSQL PL/V8 PostGIS Debian Ubuntu 2~3

A.4 FreeBSD

FreeBSD PostgreSQL <http://www.freebsd.org/ports/database.html> FreeBSD PostgreSQL FreeBSD

A.5 macOS

■ Mac 上 PostgreSQL EnterpriseDB ■ BigSQL 等の
 各種データベースをインストールする Mac 上
 Postgres. app ■ Heroku 等のクラウドサービス
 MacPorts ■ Fink 等のパッケージ管理ツール
 Mac 上 PostgreSQL EnterpriseDB ■ StackBuilder 等の
 データベース構築ツール

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □

- EnterpriseDB 提供 macOS 的 PostgreSQL 的
pgAdmin 和 StackBuilder 的
BigSQL 的 64 的 macOS
BigSQL 的 pgc 的 pgDevops 的 Web 的
A.1 的
BigSQL 的 Windows 的 PL/V8 的
- Homebrew 的 macOS 的 PostgreSQL 的
“PostgreSQL, Homebrew, and You”
Homebrew 的 PostgreSQL 的 Homebrew
PostgreSQL Wiki 的
- 的 Heroku 的 PostgreSQL.app 的
Mac 的 PostgreSQL 的
PostgreSQL 的 PostGIS 的 PL/Python 的
PL/V8 的 Postgres.app 的
- MacPorts 的 macOS 的
Mac 的 PostgreSQL 的
- Fink 的 macOS 的 Debian 的 apt-get 的

❏ B PostgreSQL❏❏❏❏❏❏❏

PostgreSQL

B.1 pg_dump

[illegible]

pg dump

[illegible]

| | |
|---------------------------------|---|
| -E, --encoding=ENCODING | ENCODING |
| -n, --schema=SCHEMA | schema |
| -N, --exclude-schema=SCHEMA | schema |
| -o, --oids | OID |
| -O, --no-owner | |
| -s, --schema-only | schema |
| -S, --superuser=NAME | |
| -t, --table=TABLE | |
| -T, --exclude-table=TABLE | |
| -x, --no-privileges | grant/revoke |
| --binary-upgrade | |
| --column-inserts | INSERT |
| --disable-dollar-quoting | SQL |
| --disable-triggers | |
| --enable-row-security | |
| --exclude-table-data=TABLE | |
| --if-exists | IF EXISTS |
| --inserts | INSERT COPY |
| --no-publications | |
| --no-security-labels | |
| --no-subscriptions | |
| --no-synchronized-snapshots | |
| --no-tablespaces | |
| --no-unlogged-table-data | WAL |
| --quote-all-identifiers | |
| --section=SECTION | pre-data data post
-data data
post-data
pre-data |
| --serializable-deferrable | |
| --snapshot=SNAPSHOT | |
| --strict-names | /schema |
| --use-set-session-authorization | SESSION AUTHORIZATION ALTER
OWNER |
| | |
| | |
| -d, --dbname=DBNAME | |
| -h, --host= | |
| -p, --port= | |
| -U, --username= | |
| -w, --no-password | |
| -W, --password | |
| --role=ROLENAME | SET ROLE |

123 PostgreSQL 10

④ PostgreSQL 9.6 安装指南

⑤⑥ PostgreSQL 9.5 安装指南

⑦ PostgreSQL 9.4 安装指南

B.2 安装指南之pg_dumpall

pg_dumpall 是 PostgreSQL 数据库系统提供的一个 SQL 脚本，用于备份整个数据库系统。B-2 节介绍了 pg_dumpall 脚本的使用方法。pg_dumpall 脚本在 2.7.2 节中

图 B-2 pg_dumpall 脚本

```
pg_dumpall --help
```

```
pg_dumpall 是 PostgreSQL 数据库系统提供的一个 SQL 脚本，用于备份整个数据库系统。
```

```
用法
```

```
pg_dumpall [选项]...
```

```
选项
```

```
-f, --file=FILENAME
```

```
输出文件
```

```
-v, --verbose
```

```
详细输出
```

```
-V, --version
```

```
显示版本信息
```

```
--lock-wait-timeout=TIMEOUT
```

```
锁等待超时时间
```

```
-, --help
```

```
显示帮助信息
```

```
选项
```

```
-a, --data-only
```

```
只输出 schema
```

```
-c, --clean
```

```
在创建 schema 之前先删除旧 schema
```

```
-g, --globals-only
```

```
只输出全局对象
```

```
-o, --oids
```

```
输出 OID
```

```
-O, --no-owner
```

```
不输出所有者信息
```

```
-r, --roles-only
```

```
只输出角色信息
```

```
-s, --schema-only
```

```
只输出 schema 名称
```

```
-S, --superuser=NAME
```

```
指定超级用户名称
```

```
-t, --tablespaces-only
```

```
只输出表空间信息
```

```
-x, --no-privileges
```

```
不输出 grant/revoke
```

```
--binary-upgrade
```

```
二进制升级
```

```
--column-inserts
```

```
使用 INSERT 语句插入数据
```

```
--disable-dollar-quoting
```

```
禁用 PostgreSQL 的美元符号引用
```

```
--disable-triggers
```

```
禁用触发器
```

```
--inserts
```

```
使用 INSERT 语句插入 COPY 数据
```

```

--no-publications                ①
--no-security-labels             ②
--no-subscriptions               ③
--no-sync                        ④
--no-security-labels
--no-tablespaces
--no-unlogged-table-data        WAL
--no-role-passwords
--quote-all-identifiers
--use-set-session-authorization SET SESSION AUTHORIZATION
ALTER OWNER

-d, --dbname=CONNSTR
-h, --host=
-l, --database=DBNAME
-p, --port=
-U, --username=
-w, --no-password
-W, --password
--role=ROLENAME

-f/--file SQL

```

①②③④ PostgreSQL 10

B.3 database pg_restore

pg_restore pg_dump PostgreSQL TAR
B-3 pg_restore
pg_restore 2.7.3

B-3 pg_restore

```

pg_restore --help

pg_restore pg_dump PostgreSQL
pg_restore [...]... [ ]

-d, --dbname=NAME
-f, --file=

```

| | |
|---------------------------------|---|
| -F, --format=c d t | 指定出力フォーマット |
| -l, --list | リストアップ |
| -v, --verbose | Verbose |
| -V, --version | バージョン |
| -, --help | ヘルプ |
| オプション | |
| -a, --data-only | スキーマのみ |
| -c, --clean | スキーマをクリーンアップ |
| -C, --create | スキーマを作成 |
| -e, --exit-on-error | エラー発生時に終了 |
| -I, --index=NAME | インデックス名 |
| -j, --jobs=NUM | ジョブ数 |
| -L, --use-list=FILENAME | スキーマリストファイル名 |
| -n, --schema=NAME | スキーマ名 |
| -N, --exclude-schema=NAME | スキーマ名を除外 ❶ |
| -O, --no-owner | 所有者を指定しない |
| -P, --function=NAME(args) | 関数名 |
| -s, --schema-only | スキーマのみ |
| -S, --superuser=NAME | スーパーユーザー名 |
| -t, --table=NAME | テーブル名 ❷ |
| -T, --trigger=NAME | トリガー名 |
| -x, --no-privileges | 権限をgrant/revokeしない |
| -l, --single-transaction | 単一トランザクション |
| --enable-row-security | 行セキュリティを有効にする ❸ |
| --disable-triggers | トリガーを無効にする |
| --no-data-for-failed-tables | 失敗したテーブルにデータを付与しない |
| --no-publications | パブリケーションを無効にする ❹ |
| --no-security-labels | セキュリティラベルを無効にする |
| --no-subscriptions | サブスクリプションを無効にする ❺ |
| --no-tablespaces | テーブルスペースを無効にする |
| --section=SECTION | スキーマのセクション: pre-data, data, post-data, data, post-data, pre-data, post-data |
| --strict-names | スキーマ名/テーブル名を厳格に ❻ |
| --use-set-session-authorization | SET SESSION AUTHORIZATIONを使用 |
| ALTER | OWNERを指定 |
| 接続オプション | |
| -h, --host=HOST | ホスト名 |
| -p, --port=PORT | ポート番号 |
| -U, --username=USER | ユーザー名 |
| -w, --no-password | パスワードを指定しない |

```
-W, --password  
--role=ROLENAME
```

データベースに接続する
データベースSET ROLE

①②③ PostgreSQL 10 をインストール

④⑤ PostgreSQL 9.6 をインストール 9.6 をインストール - t をインストール 9.6 をインストール

⑥ PostgreSQL 9.5 をインストール

B.4 psql

B-4 psql をインストール 3.1 3.2 をインストール

B-4 psql をインストール

```
\?  
データベース  
\copyright          PostgreSQLデータベース  
\errverbose         データベース ①  
\g [DB] or ;       データベース | データ  
\gexec             データベース ②  
\gset [PREFIX]     データベース psql  
\h [DB]            SQLデータベース * データベース  
\gx [DB]           データ \g データベース ③  
\q                データ psql  
\crosstabview [COLUMNS] データベース ④  
\watch [SEC]       データ SEC  
  
データベース  
  \? [commands]     データベース  
  \? options        データ psql データベース  
  \? variables      データベース  
  \h [DB]           SQLデータベース * データベース  
  
データベース  
\e [FILE] [LINE]    データベース  
\ef [FUNCNAME [LINE]] データベース  
\ev [VIEWNAME [LINE]] データベース ⑤  
\p                 データベース  
\r                 データベース  
\w [DB]            データベース
```

SQL/PLSQL

\copy ...

\echo [text]

\i [file]

\ir FILE

\o [file]

\qecho [text]

PL/SQL ⑥

\if EXPR

\elif EXPR

\else

\endif

PL/SQL

PL/SQL S = PL/SQL + PL/SQL

\d[S+] [table]

\d[S+] [table]

\da[S] [table]

\dA[+] [table]

\db[+] [table]

\dc[S] [table]

\dC [table]

\dd[S] [table]

\ddp [table]

\dD[S] [table]

\det[+] [table]

\des[+] [table]

\deu[+] [table]

\dew[+] [table]

\df[antw][S+] [table]

\dF[+] [table]

\dFd[+] [table]

\dFp[+] [table]

\dFt[+] [table]

\dg[S+] [table]

\di[S+] [table]

\dl

\dL[S+] [table]

\dm[S+] [table]

\dn[S+] [table]

\do[S] [table]

\dO[S+] [table]

\dp [table]

SQL COPY PL/SQL

PL/SQL

PL/SQL

\i PL/SQL

PL/SQL

PL/SQL | PL/SQL

PL/SQL \echo PL/SQL

PL/SQL \o PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL ⑦

PL/SQL

PL/SQL conversion PL/SQL

PL/SQL cast PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL a-PL/SQL/n-PL/SQL/t-PL/SQL

/w-PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL

PL/SQL \lo_list PL/SQL

PL/SQL

PL/SQL

PL/SQL schema PL/SQL

PL/SQL

PL/SQL

PL/SQL

| | |
|-------------------|----------|
| \drds [001 [002]] | database |
| \dRp[+] [PATTERN] | 8 |
| \dRs[+] [PATTERN] | 9 |
| \ds[S+] [00] | |
| \dt[S+] [00] | |
| \dT[S+] [00] | |
| \du[S+] [00] | |
| \dv[S+] [00] | |
| \dE[S+] [00] | |
| \dx[+] [00] | |
| \dy [00] | |
| \l[+] | |
| \sf[+] FUNCNAME | |
| \sv[+] VIEWNAME | 10 |
| \z [00] | \dp |

| | |
|--------------------------|---------------------------------------|
| \a | |
| \C [000] | |
| \f [000] | |
| \H | HTML |
| \pset NAME [VALUE] | NAME format border expanded |
| fieldsep | fieldsep_zero footer null |
| numericlocale | recordsep tuples_only title tableattr |
| pager | pager_min_lines recordsep |
| recordsep_zero tableattr | title tuples_only |
| unicode_border_linestyle | unicode_column_linestyle |
| unicode_header_linestyle | |
| \t [on off] | |
| \T [000] | HTML |
| \x [on off] | |

| | |
|---|----------|
| \c[onnect] {[DBNAME - USER - HOST - PORT -] conninfo} | postgres |
| \encoding [0000] | |
| \password [USERNAME] | |
| \conninfo | |

| | |
|----------------------|--|
| \cd [00] | |
| \setenv NAME [VALUE] | |

```
\timing [on|off]           実行時間を出力する
! [ ]                     shell環境で実行する
```

❶❷❸ PostgreSQL 10 のインストール

❹❺❻��❽❾❿ PostgreSQL 9.6 のインストール

❶ PostgreSQL 9.5 のインストール

B.5 psqlのインストール

❶ B-5 のインストール psql のインストール 3.2

❶ B-5 psql のインストール

```
psql --help

psql PostgreSQL のコマンドラインインタプリタ
psql [ ]... [database ] [ ]

-c, --command=      SQLコマンドを実行する
-d, --dbname=       データベース名
-f, --file=         ファイル名
-l, --list           データベース名をリストアップ
-v, --set=, --variable=NAME=VALUE  環境変数NAMEをVALUEで設定
                                (psqlはON_ERROR_STOP=1)
                                (psqlrcは~/psqlrc)
-X, --no-psqlrc     設定ファイルpsqlrcを読み込まない
-l ("one"), --single-transaction  単一トランザクションモード
-?, --help[=options]
    --help=commands  コマンドに関するヘルプ
    --help=variables  変数に関するヘルプ
--version           バージョン情報を表示

-a, --echo-all      実行結果をすべて表示
-b, --echo-errors    エラーメッセージを表示
-e, --echo-queries   実行されるSQL文を表示
-E, --echo-hidden    隠れたメッセージを表示
-L, --log-file=      ログファイル名
-n, --no-readline    readlineライブラリを使用しない
```



```

-o, --output=FILENAME          指定輸出檔案名稱 | 指定
-q, --quiet                    靜音模式
-s, --single-step              單步模式
-S, --single-line              單行SQL模式

選項
-A, --no-align                 不對齊
-F, --field-separator=FIELD_SEPARATOR 指定欄位分隔符號 | 指定
-H, --html                     HTML輸出
-P, --pset=VAR[=ARG]           指定VAR變數的ARG值 \pset 指定
-R, --record-separator=RECORD_SEPARATOR 指定記錄分隔符號
-t, --tuples-only              只輸出元組
-T, --table-attr=TABLE_ATTR    HTML輸出時指定表格屬性
-x, --expanded                 展開輸出
-z, --field-separator-zero     欄位分隔符號為零
-0, --record-separator-zero    記錄分隔符號為零

選項
-h, --host=HOST                指定主機名稱
-p, --port=PORT                指定連接埠 (預設為5432)
-U, --username=USERNAME        指定使用者名稱
-w, --no-password              不輸入密碼
-W, --password                 輸入密碼

在psql中輸入 \? 或 \help 可顯示SQL命令清單 PostgreSQL 命令
psql

```

❶❷❸ PostgreSQL 9.5 安裝與設定

❶❷❸

Regina Obe 著 • 譯者：Paragon Corporation 20 年 PostgreSQL 專家
 本書介紹了 PostgreSQL 9.5 的安裝與設定，並介紹了 PostGIS、pgRouting 和 GEOS 等空間數據庫工具。
PostGIS in Action 和 *pgRouting: A Practical Guide* 也是相關書籍。

Leo Hsu 著 • 譯者：Paragon Corporation 20 年 PostgreSQL 專家
 本書介紹了 PostgreSQL 9.5 的安裝與設定，並介紹了 PostGIS、pgRouting 和 GEOS 等空間數據庫工具。

Leo PostGIS in Action pgRouting: A Practical Guide

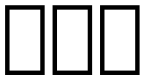
Macroscelides proboscideus

45 60 1 3

5 15 41 46

O'Reilly animals.oreilly.com

Meyers Kleines



contact@turingbook.com

ebook@turingbook.com

- @ :
- @ :
- @ :
- : ituring_interview
- : turingbooks

091507240605ToBeReplacedWithUserId